2016-06

Automated network mapping and topology verification

Collier, Anthony R.

Monterey, California: Naval Postgraduate School

http://hdl.handle.net/10945/49438

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

AUTOMATED NETWORK MAPPING AND TOPOLOGY VERIFICATION

by

Anthony R. Collier

June 2016

Thesis Advisor:	Gurminder Singh
Co-Advisor:	John Gibson

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2016	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE AUTOMATED NETWORK MAPPING AND TOPOLOGY VERIFICATION	**5. FUNDING NUMBERS**
6. AUTHOR(S) Anthony R. Collier	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The current military reliance on computer networks for operational missions and administrative duties makes network stability and security a high priority for military units. The rapid rate at which technology changes means that constant and continued training is required for the skilled professionals who maintain and secure these computer networks. Current training methods are insufficient at representing the complex nature of the typical modern military computer network and the continually evolving nature of the attacks to networks. The Mapping, Awareness, and Virtualization Network Administrator Training Tool (MAVNATT) is a proposed system designed to replicate operational computer networks, through virtualization, providing a stable, accurate, and safely partitioned training environment that can closely mimic the configuration and functionality of any operational network. This research provides a solution for the mapping module of the MAVNATT system in the form of an application. During testing, we successfully developed network plans, visualized and verified those plans, scanned live networks for comparison and validation against those plans, and exported the network configurations for import by the MAVNATT awareness and virtualization modules. The mapping application was developed on a foundational framework that facilitates expansion and increased functionality during future research.

14. SUBJECT TERMS network mapping, network verification, network topology, network administrator training, MAVNATT, MAST, tactical network	**15. NUMBER OF PAGES** 97
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

THIS PAGE INTENTIONALLY LEFT BLANK

AUTOMATED NETWORK MAPPING AND TOPOLOGY VERIFICATION

Anthony R. Collier
Captain, United States Marine Corps
B.A., The University of Texas, 2009

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2016

Approved by: Gurminder Singh
Thesis Advisor

John Gibson
Co-Advisor

Peter Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The current military reliance on computer networks for operational missions and administrative duties makes network stability and security a high priority for military units. The rapid rate at which technology changes means that constant and continued training is required for the skilled professionals who maintain and secure these computer networks. Current training methods are insufficient at representing the complex nature of the typical modern military computer network and the continually evolving nature of the attacks to networks. The Mapping, Awareness, and Virtualization Network Administrator Training Tool (MAVNATT) is a proposed system designed to replicate operational computer networks, through virtualization, providing a stable, accurate, and safely partitioned training environment that can closely mimic the configuration and functionality of any operational network. This research provides a solution for the mapping module of the MAVNATT system in the form of an application. During testing, we successfully developed network plans, visualized and verified those plans, scanned live networks for comparison and validation against those plans, and exported the network configurations for import by the MAVNATT awareness and virtualization modules. The mapping application was developed on a foundational framework that facilitates expansion and increased functionality during future research.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

ARP	address resolution protocol
CIDR	classless inter-domain routing
CSV	comma-separated values
DNS	domain name system
ICMP	Internet control message protocol
IETF	Internet engineering task force
IP	Internet protocol
JUNG	Java universal network/graph
LAN	local area network
MAC	media access control
MAVNATT	mapping, awareness, and virtualization network administrator training tool
NIC	network interface card
OS	operating system
OSI	open systems interconnection
RFC	request for comments
TCP	transmission control protocol
UML	unified modeling language
XML	extensible markup language

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank all of my friends and family who have given me encouragement and support over the years. To my silly, smart, and beautiful daughters, Grace and Emily: you always make me smile. Thank you for being so amazing. Most of all, I would like to thank my lovely wife, Nicole. You have helped keep me on track by providing countless hours of encouragement, motivation, and support. You have endured long restless nights, numerous deadlines, and lots of “grumpy Anthony” as a result of this research. I cannot wait to start the rest of our lives together.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE

Today's military is increasingly reliant on technology and connectivity to conduct training, day-to-day operations, and mission execution. Computer networks have become ubiquitous at all levels of organization, including the small units that make up the foundation of the Marine Corps. These networks are a cornerstone for modern operations and communication: they must be secure and robust. The requisite proper maintenance and administration of military computer networks is a challenging task, which requires continual training to stay abreast with the pace of changing technology and the periodic emergence of external threats.

Due to the inherent nature and complexity of computer networks, variations in configuration and topology are commonplace. For this reason, the live networks found at the small-unit level often present a unique set of security and administration requirements. Training scenarios executed on a live network provide the best opportunity for realism and functionality; however, doing so introduces additional risks and the potential for operational impact should a network problem arise during training. For this reason, network administrators are often required to seek alternative solutions for network training and system testing.

A typical model for network administrator training involves the establishment of a small, stand-alone network using physical hardware. Users can adjust settings, introduce changes or malicious programs, and conduct network-related simulations and scenarios using this training network. The setup and configuration of such networks is laborious, and the size and scope of these networks are necessarily limited by available space and equipment. Creating an exact working physical replica of a live network that correctly mimics traffic and topology is not a viable option. These factors limit the utility of training on such networks.

The Mapping, Awareness, and Virtualization Network Administrator Training Tool (MAVNATT) [1] is a proposed system that provides a solution to deficiencies in the current training model. MAVNATT is a tool that allows network administrators to plan and validate a network layout, scan a live network to compare configuration settings against the plan, monitor the live network for changes or the emergence of potential threats, and create a virtualized replica of the network. This virtualized version of a network can be used for training and evaluation purposes, allowing network administrators to train on an isolated network "virtually" identical to their live network.

The focus of this thesis is the development of an application that satisfies the requirements for the mapping portion of the MAVNATT system. This application should provide functionality for network administrators to develop a network plan, visualize and verify that plan, scan the live network for comparison and validation, and export the network configuration for import and use by the MAVNATT awareness and virtualization modules.

B. SCOPE AND BOUNDARIES

The mapping application detailed in this thesis is intended to be used as a tool by network administrators to accurately and efficiently capture details of their LAN configuration and topology. These details can be stored or exported for use in other applications within the MAVNATT system. There is a large amount of information available to categorize and describe a network, making it necessary to limit the scope of and boundaries of mapping efforts. This implementation of the MAVNATT mapping application makes the following assumptions:

- The network administrator using the application has system administrator-level privileges for the respective operational network and is able to install and execute programs and send traffic over that network.
- The user has prior knowledge of the network that is to be mapped, including the expected configuration and details of individual connected devices.
- Only devices on the local network are to be included in the results of the mapping effort. External devices will not be mapped.
- The mapping application does not consider physical network topology, meaning the actual physical location of devices on the network.

- Logical and hardware addresses of individual devices on the network are static and are not expected to change after their initial assignment.
- The mapping application provides a snapshot of the current network configuration at the time of execution.

C. THESIS ORGANIZATION

This thesis is organized into four additional chapters: background, design and implementation, testing and evaluation, and conclusion and proposed future work.

1. Chapter II: Background

This chapter describes the purpose and characteristics of the proposed MAVNATT system along with some defining characteristics and properties of local area networks (LANs). Detailed analysis of LAN traffic provides all the information necessary to accurately map a network's topology and catalog individual devices connected to that network. The chapter also gives a brief description and analysis of some existing tools and programs that can be used to analyze and evaluate relevant network traffic.

2. Chapter III: Design and Implementation

This chapter describes the design and implementation of the MAVNATT mapping application, as well as the program flow of execution when using it to map a network. The program architecture and framework are described in detail, allowing for further implementation and adaptation during future research. This object-oriented framework allows for the capture and manipulation of all necessary data to accurately describe a network, individual attached devices, and the complex relational interaction between devices. The program flow of execution is a detailed step-by-step walkthrough of the mapping process from start to finish.

3. Chapter IV: Testing and Evaluation

This chapter describes testing and evaluation of the mapping application, to include results and findings. Individual tests are conducted to evaluate the mapping application's ability to represent a planned network while identifying potential errors in the respective network plan. Live network scans are conducted on a controlled physical network environment, allowing testing and evaluation of specific network details. The network environment is configured to closely resemble a typical topology of networks used by small units in the Marine Corps.

4. Chapter V: Conclusion and Future Work

The final chapter provides a summary of work conducted in this thesis, its findings, and the resulting functionality of the MAVNATT mapping application. Suggestions for areas of future work and research allow for continued development, integration and implementation of the mapping application, and the MAVNATT system as a whole.

II. BACKGROUND

The proper design and configuration of a computer network is a complex task. Simple mistakes in configuration files or device settings can lead to network instability, unexpected network activity, or the inability to use the network as intended. Verification of these settings can be accomplished through analysis of data as it transits the network. However, the sheer volume of available data makes manual network verification tedious and prone to errors. Automated tools allow network administrators to more easily gather and analyze data and develop a picture of the proposed and existing network topology. A thorough definition of the term “network topology” further facilitates the proper use of available network data to make inferences and conclusions about the layout and configuration of that network, including number and role of comprising elements, along with their representative data patterns. Once these are properly scoped, we can use existing tools to automate traffic analysis and effectively map a network.

A. MAVNATT

MAVNATT is a system proposed by Naval Postgraduate School student, Daniel McBride [1], which is undergoing continued research and development. McBride identified challenges presented to military network administrators when establishing and maintaining small-scale networks, such as those found at the individual unit level. Upon properly configuring a computer network, the system administrator often forbids any changes to that network for fear of disrupting the stable state. This inaccessibility to the live network limits options for training scenarios. The current model is to establish a separate stand-alone network on which users can introduce problems or scenarios for training purposes.

The MAVNATT system (see Figure 1) allows a user to replicate a live network in a virtual environment that can then be used for training. The mapping module provides functionality to scan and accurately represent an existing live network and is the focus of this thesis. The awareness module provides the network administrator with real-time updates of changes and status of the network. The Virtualization module generates a replica of the live network in a virtualized environment. All modules are interconnected by the underlying framework, and together they can be used to both monitor the live network and provide a realistic training environment.

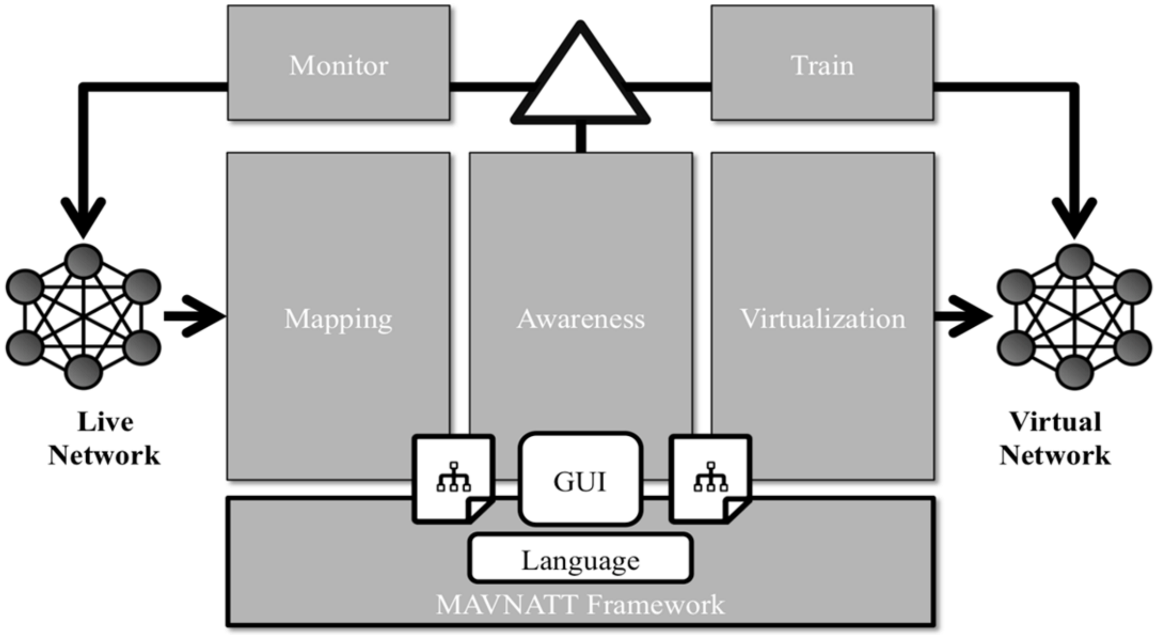
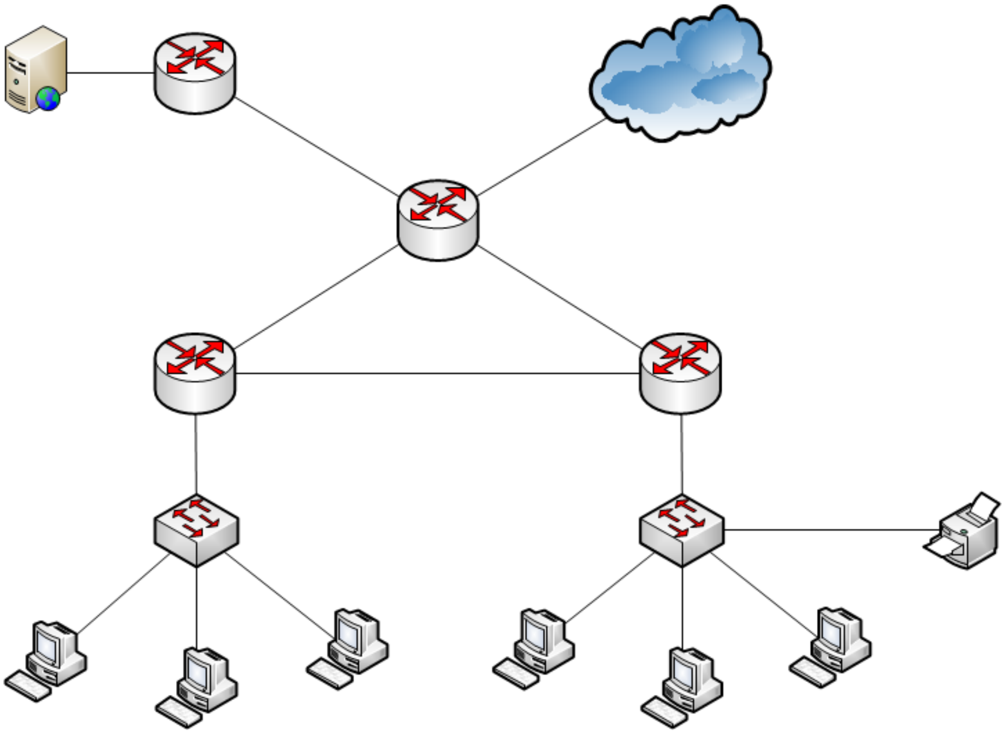


Figure 1. Proposed MAVNATT Framework. Source: [1].

The MAVNATT model allows for more effective training than the current model, as it simulates operation on the actual live network with duplicate configurations and settings. The focus of this thesis is the development of the mapping module, which will import a proposed network plan, scan the live network, and provide specifications of either the planned network or the actual network. The virtualization module can use these specifications to create an accurate replica of the network for awareness and training purposes.

B. NETWORK TOPOLOGY AND CHARACTERISTICS

We define network topology as the collection of information detailing the logical layout and configuration of networked devices, including the interconnections between those devices (see Figure 2). This collection of information includes amplifying data about the networked devices such as hardware details, logical addressing schemes, operating system used, applications in use, and potential user information. Devices such as routers, switches, and hubs facilitate forwarding traffic throughout the local network and, specific to the routers, beyond. Client devices are the end users and can include computers, printers, hand-held devices; essentially any device that can be connected to the network to send and receive data. In this thesis, we focus on the topology of LANs. In addition to connected devices, the subdivision of the LAN into even smaller subnetworks is an important characteristic of network topology and must be given due consideration, as it greatly complicates the mapping task.



A typical LAN topology distributes client devices and locally hosted servers across multiple subnets, allowing for greater control of traffic flow.

Figure 2. Example LAN Configuration.

It is important to note that we are not concerned with physical topology, meaning the physical location of devices and their relative distance to other devices in the network. Rather, we are concerned with the logical layout: which routers are connected to one another and at what points are clients connected to the network.

1. Open Systems Interconnection Reference Model

The network topology can be derived through careful analysis of existing traffic on the network. However, the type and composition of traffic accessible on a LAN varies widely depending on the connected devices and the applications executing on those devices. The Open Systems Interconnection (OSI) Reference Model is an abstract concept of data encapsulation that allows for the transit of data through the internal structure of a computer as well as across networks [2]. This layered approach of abstraction (see Figure 3) facilitates interfacing between applications and hardware without the requirement to design and implement a system from the ground up for each new application. The model also allows for the scaling of systems and networks while ensuring compatibility between networking entities [2]. While the OSI Model outlines an overall concept for the design of networked systems, it does not detail any concrete protocols or specifications. These specifics are left to other models that implement the OSI Model guidelines.

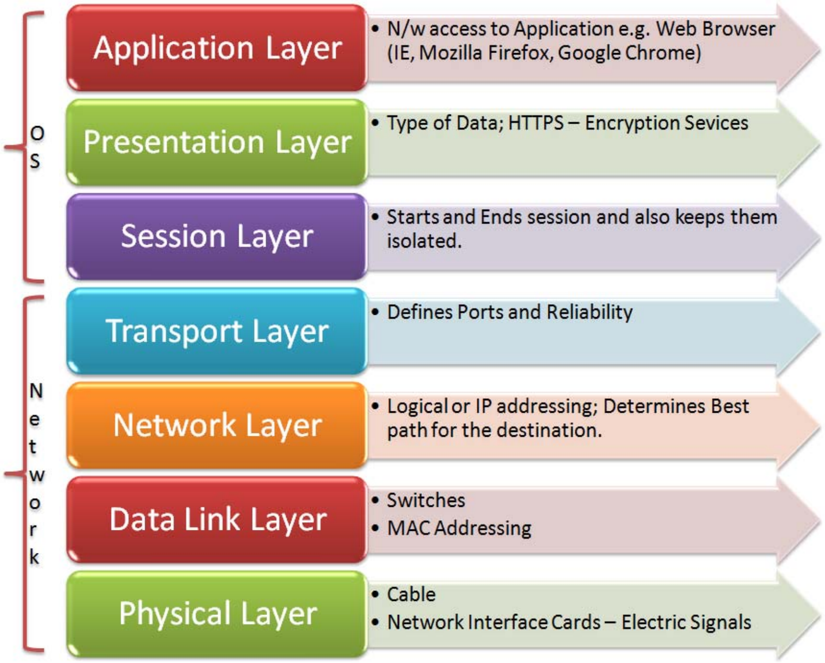

OS
Network
Application Layer
• N/w access to Application e.g. Web Browser (IE, Mozilla Firefox, Google Chrome)
Presentation Layer
• Type of Data; HTTPS – Encryption Sevices
Session Layer
• Starts and Ends session and also keeps them isolated.
Transport Layer
• Defines Ports and Reliability
Network Layer
• Logical or IP addressing; Determines Best path for the destination.
Data Link Layer
• Switches
• MAC Addressing
Physical Layer
• Cable
• Network Interface Cards – Electric Signals

Figure 3. Overview of the OSI Reference Model. Source: [3].

a. Physical Layer

The physical layer describes the physical medium on which data signals are actually transmitted. This medium is the underlying architecture of any connected network and it is on this layer that raw digital or analogue traffic transits from one node to the next. While there can be some information gleaned from this raw data, analysis of transmissions on the physical layer is outside of the scope of this thesis. However, the actual capture of network traffic requires some attachment to the physical layer by which the upper layer information is extracted. This thesis assumes such access is available.

b. Data Link Layer

The data link layer is the first layer above the physical layer and it describes how individual pieces of hardware encapsulate data before the actual bits are transmitted to the next node. Data to be transmitted is grouped into protocol data units called frames and encapsulated with header and trailer metadata, allowing devices to determine the next-hop destination of the frame and take action accordingly. The device hardware address, a globally unique identifying number for a network interface, is included in this metadata [4]. The section of a network in which devices can communicate directly with each other via the data link layer is referred to as a broadcast domain. The hardware address is carried on all frames of traffic within a single broadcast domain only and is not translated or transferred when a packet transits through a router to another broadcast domain. This localization of the hardware address is a key piece of information that can be used to classify network traffic and identify connected devices during analysis.

c. Network Layer

The network layer contains valuable information for determining the logical topology and layout of a network. Here, frames are encapsulated with more metadata that includes information about the logical addressing scheme, most commonly the Internet Protocol (IP). Logical addressing allows traffic to be routed across broadcast domains and is an important detail of a LAN’s topology, with each device requiring a unique logical address in order to send and receive traffic correctly. In-depth information regarding the allocation of addresses among routers and hosts is vital to gaining a complete understanding of the network. Knowledge of a subnet allows us to classify which hosts are a part of the network and make a determination about how many broadcast domains exist.

Routers use manually or automatically configured routing tables to forward each packet to the next hop (egress point of the connected broadcast domain) within a route from a source to destination. The range of possible IP addresses is sub-divided and allocated across all users of the Internet. Additionally, private IP ranges can be utilized within LANs, as defined by Request for Comments (RFC) 1918 [5], though they must undergo an address translation before traffic from the LAN can be routed outside of that LAN. Every broadcast domain is represented by a single subnet, or subdivision of IP addresses.

For the purposes of this research, we characterize groups of IP addresses (subnets) using the Classless Inter-Domain Routing (CIDR) scheme and identify a subnet by its subnet address and subnet mask [6]. The subnet address is the first IP address in the range allocated for that subnet. The subnet mask is a bit string that delineates which bits in an address designate the network and which designate a host, as depicted in Figure 4. Note that this bit string is a set of left-justified 1-bits corresponding to the significant bits of the address that represent the network, with the remaining bits of the string set to “0,” corresponding to the bits that form the unique identified for the entities connected to the subnet. Thus, all IP addresses in a single subnet will have the same network bits in the address. Given a particular IP address and the subnet scheme, we can determine to which broadcast domain the IP address belongs [6].

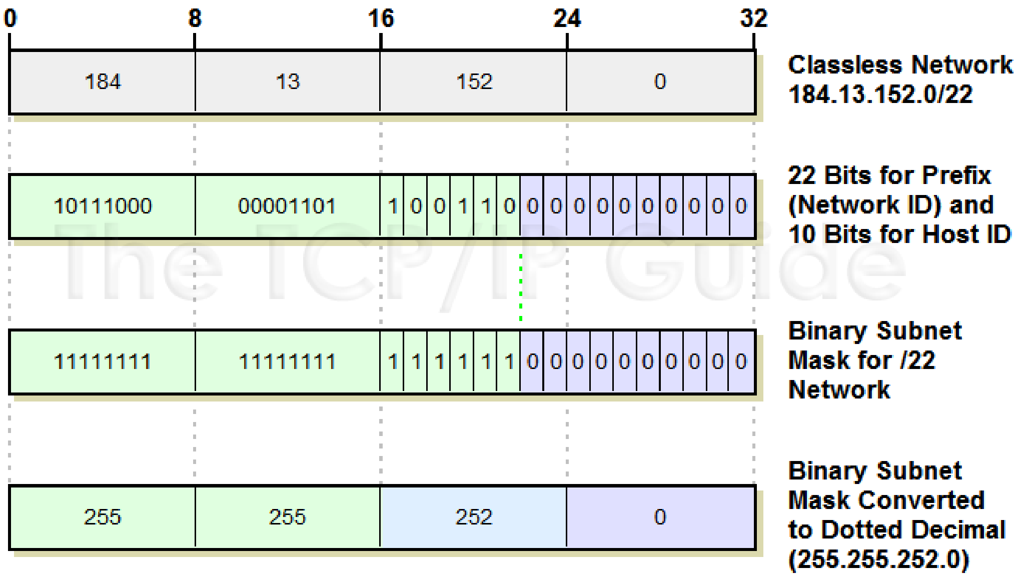

0
8
16
24
32
184
13
152
0
Classless Network 184.13.152.0/22
10111000
00001101
1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
22 Bits for Prefix (Network ID) and 10 Bits for Host ID
11111111
11111111
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
Binary Subnet Mask for /22 Network
255
255
252
0
Binary Subnet Mask Converted to Dotted Decimal (255.255.252.0)
The TCP/IP Guide

Figure 4. Example Subnet Addressing Using CIDR Notation. Source: [7].

d. Transport Layer

The physical, data link, and network layers are similar in that they are all involved in the actual delivery of data from one node on the network to another. Conversely, the session, presentation, and application layers are where data is actually generated [8]. The transport layer handles the transition from the upper layers to the lower layers with the chief function of managing application data and preparing it for exchange between the communicating application entities. Additionally, as incoming data is received from the network layer, services running on the transport layer manage forwarding that data to the appropriate application. This allows for many applications and services to run on the same computer at once with a single network connection.

e. Application-Level Layers

Beyond the transport layer are the session, presentation, and application layers. These layers are similar in that they all involve the actual generation and representation of data. For the purpose of this thesis, we refer to these layers as a single group, called the application-level layers. These layers are generally decoupled from the actual act of transmitting data on the network, which is handled by the bottom three layers. However, this data, referred to as the payload, is still visible on the network, and it is the reason for the traffic in the first place. While the payload usually does not contain network specific information, some applications provide functionality for network establishment and management, such as routing protocols like Border Gateway Protocol. Analysis of these packet payloads can yield an abundance of information about the devices connected to the network, adding to our depth of understanding of the topology.

2. Internet Protocol Suite

While often referred to as a reference model as well, the IP Suite is the most commonly used collection of standards and protocols pertinent to transfer of data between networking domains. It was, and continues to be, developed and maintained by the Internet Engineering Task Force (IETF) to facilitate consistency across system and application developers. These protocols and standards are explicitly applicable for functionalities above what the OSI Reference Model identifies as Physical and Data Link Layer functions (see Figure 5). Protocol specifications are published via RFCs and are continually updated to ensure completeness and security of protocols in an evolving network community [9]. These protocols are typically implemented by the various operating systems installed on the networked devices and hosts. IP and the Transmission Control Protocol (TCP) are the most commonly used protocols in the set, and therefore the IP Suite is also frequently referred to as the TCP/IP Suite. Payload and header information from the packets of some of these protocols provide the necessary information to map the topology of a network.

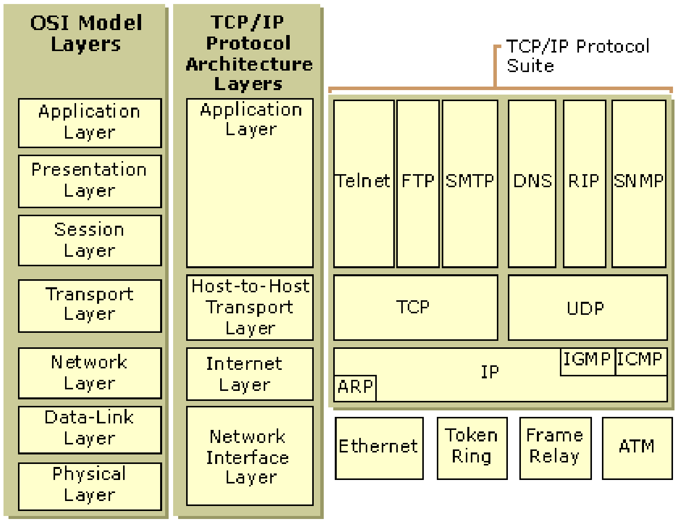


Figure 5. OSI Reference Model and the IP Suite Comparison. Source: [10].

a. Address Resolution Protocol

Address Resolution Protocol (ARP) is used by network devices to correlate a hardware address to the corresponding logical address for a given device. Most commonly, it allows devices to map Media Access Control (MAC) addresses to IP addresses. ARP technically operates to support the network layer, but packets do not cross over broadcast domains. Primary ARP messages are the ARP Request by which a device issues a query for a MAC address correlating to a specific IP address and the ARP Reply response from the queried device. The layout of individual data fields within an ARP packet is detailed in Figure 6 [11]. Note that the destination layer 3 address is the network address (e.g., IP) for which the layer 2 address is required and the destination layer 2 address in the query is the layer 2 broadcast address. For networks employing the Institute of Electrical and Electronics Engineers protocols, the layer 2 broadcat address is 48 bits, all ones (i.e., 0xFF:FF:FF:FF:FF:FF in hexadecimal format, colons added for clarity only).

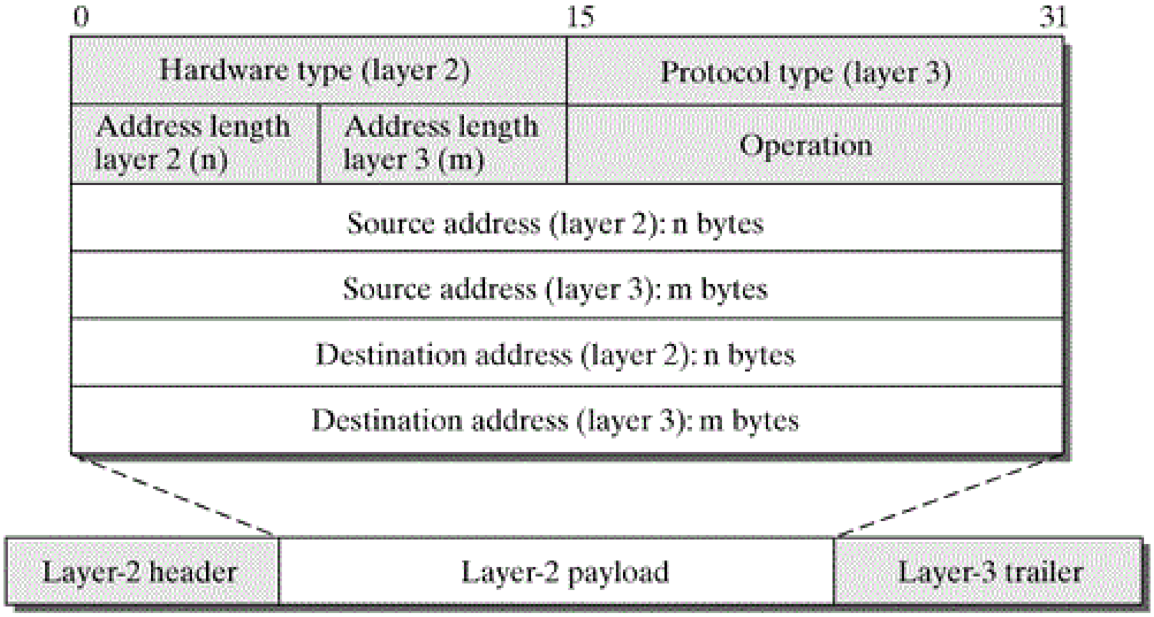
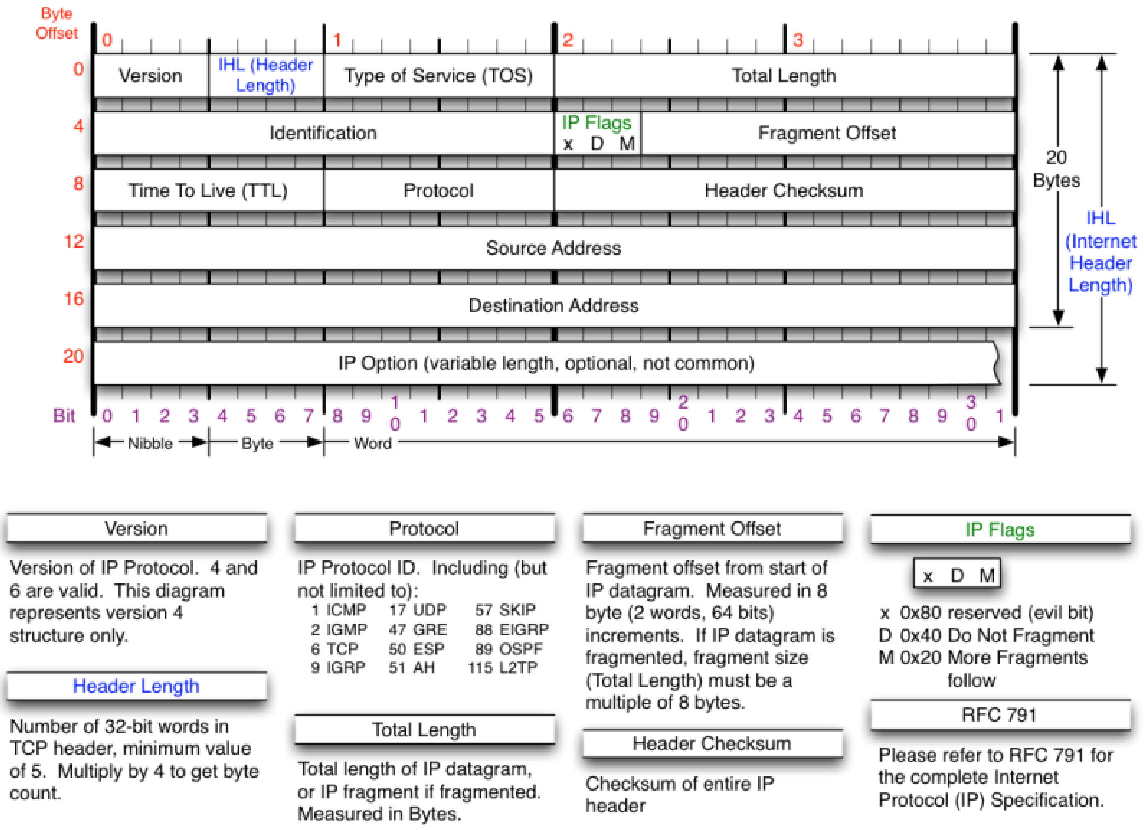

0
15
31
Hardware type (layer 2)
Protocol type (layer 3)
Address length layer 2 (n)
Address length layer 3 (m)
Operation
Source address (layer 2): n bytes
Source address (layer 3): m bytes
Destination address (layer 2): n bytes
Destination address (layer 3): m bytes
Layer-2 header
Layer-2 payload
Layer-3 trailer

Figure 6. ARP Packet Data Fields. Source: [11].

b. Internet Protocol

IP is the most commonly used protocol for assigning logical addresses to network layer devices. The information available in an IPv4 header can be very useful for mapping network topology. IPv4 and IPv6 headers differ slightly, and for the purposes of this thesis we will investigate IPv4 packets only. The overall structure of the IPv4 packet header is shown in Figure 7. Some of the useful fields in the IP header are protocol, source, and destination addresses. The protocol field contains the identifier for next layer protocol used in the data portion of the IP packet, normally the transport layer protocol, although as indicated by Figure 5 and Figure 7, it could be ARP, Internet Control Message Protocol (ICMP), or another protocol operating directly on top of IP. This information is necessary to determine how to handle the packet at the destination and how to decode the encapsulated data. Source and destination addresses are the sender and receiver of the packet. These addresses do not change for the life of the packet and remain intact as the data transits across broadcast domains [12].



Version

Version of IP Protocol. 4 and 6 are valid. This diagram represents version 4 structure only.

Header Length

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

Protocol

IP Protocol ID. Including (but not limited to):

1 ICMP	17 UDP	57 SKIP
2 IGMP	47 GRE	88 EIGRP
6 TCP	50 ESP	89 OSPF
9 IGRP	51 AH	115 L2TP

Total Length

Total length of IP datagram, or IP fragment if fragmented. Measured in Bytes.

Fragment Offset

Fragment offset from start of IP datagram. Measured in 8 byte (2 words, 64 bits) increments. If IP datagram is fragmented, fragment size (Total Length) must be a multiple of 8 bytes.

Header Checksum

Checksum of entire IP header

IP Flags

x D M

x 0x80 reserved (evil bit)
D 0x40 Do Not Fragment
M 0x20 More Fragments follow

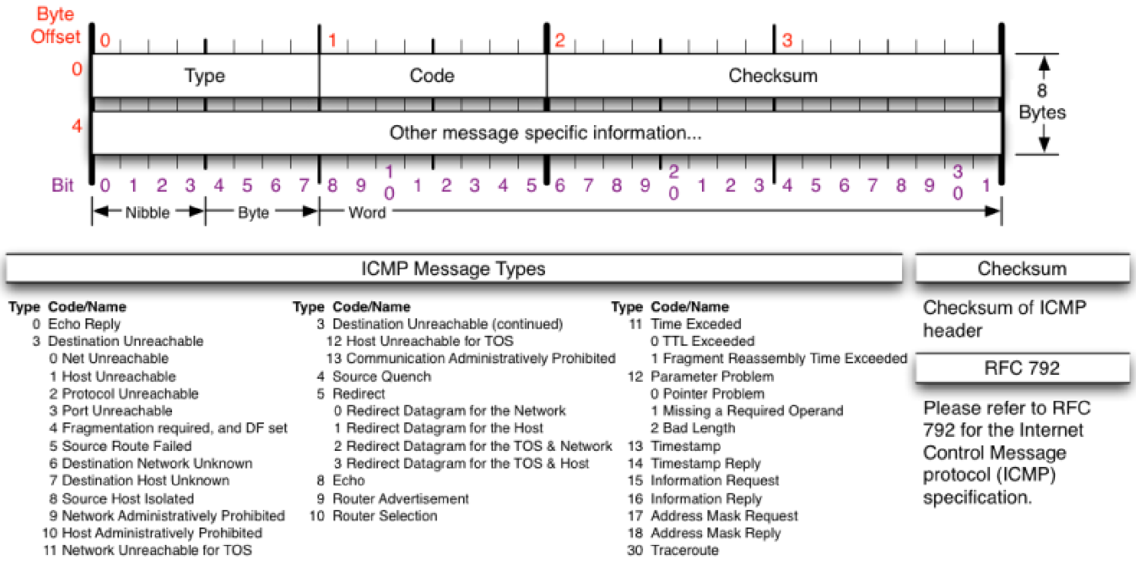
RFC 791

Please refer to RFC 791 for the complete Internet Protocol (IP) Specification.

Figure 7. IPv4 Packet Header Data Fields. Source: [13].

c. *Internet Control Message Protocol*

ICMP is a messaging protocol used to notify network devices that a problem occurred while a packet was in transit. Additionally, it is a mechanism to send network-related control messages that can be used for diagnostic purposes. ICMP header fields, shown in Figure 8, are type, code, and checksum, followed by a four-byte section that varies depending on type and code. For error messages, the data section of an ICMP packet contains a copy of the IPv4 header for the packet that caused the error, along with the first eight bytes of data from the original packet. The entire ICMP packet is then encapsulated in a new IPv4 header for proper routing to the intended recipient of the error message.



ICMP Message Types

Type	Code/Name
0	Echo Reply
3	Destination Unreachable
	0 Net Unreachable
	1 Host Unreachable
	2 Protocol Unreachable
	3 Port Unreachable
	4 Fragmentation required, and DF set
	5 Source Route Failed
	6 Destination Network Unknown
	7 Destination Host Unknown
	8 Source Host Isolated
	9 Network Administratively Prohibited
	10 Host Administratively Prohibited
	11 Network Unreachable for TOS

Type	Code/Name
3	Destination Unreachable (continued)
	12 Host Unreachable for TOS
	13 Communication Administratively Prohibited
4	Source Quench
5	Redirect
	0 Redirect Datagram for the Network
	1 Redirect Datagram for the Host
	2 Redirect Datagram for the TOS & Network
	3 Redirect Datagram for the TOS & Host
8	Echo
9	Router Advertisement
10	Router Selection

Type	Code/Name
11	Time Exceded
	0 TTL Exceeded
	1 Fragment Reassembly Time Exceeded
12	Parameter Problem
	0 Pointer Problem
	1 Missing a Required Operand
	2 Bad Length
13	Timestamp
14	Timestamp Reply
15	Information Request
16	Information Reply
17	Address Mask Request
18	Address Mask Reply
30	Traceroute

Checksum

Checksum of ICMP header

RFC 792

Please refer to RFC 792 for the Internet Control Message protocol (ICMP) specification.

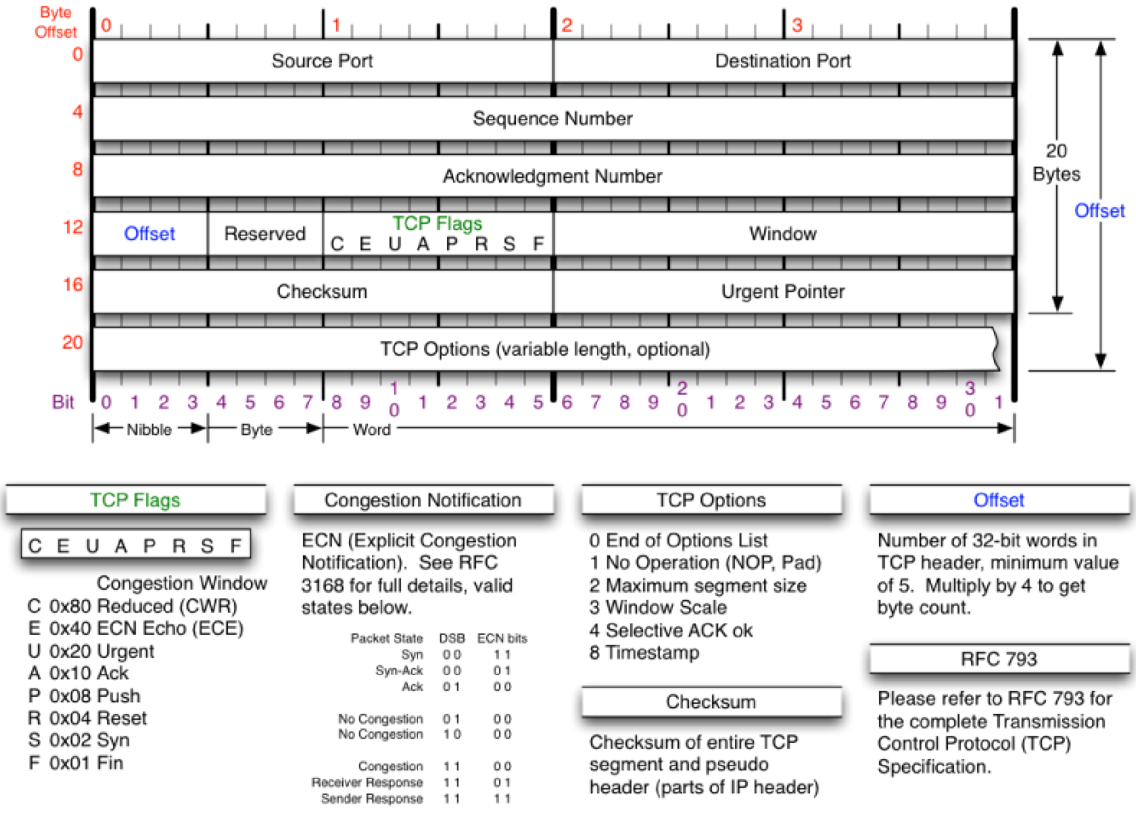
Figure 8. ICMP Packet Header Data Fields. Source: [13].

Echo request and echo reply packets perform a function commonly referred to as a ping, a diagnostic tool in which one device sends a short request for response to a remote device. The remote device issues an echo reply. This ping functionality can be used to test proper network connectivity and latency between devices.

The destination unreachable error message is generated either at the end device, the last-hop router before that end device, or by a router that does not have a path to the destination network. This error is generated in response to a destination port, device, or network being unreachable. This functionality is useful for probing a network for the existence of a particular device or for scanning a device to determine which ports are open [14].

d. *Transmission Control Protocol*

TCP is a transport-layer protocol that provides for the connection-based transfer of data between two networked devices. The TCP header, shown in Figure 9, encapsulates the application data [15]. TCP correlates to the transport and session layers of the OSI Model with the primary purpose of establishing a host-to-host connection between two networked devices. Once this connection is established, the functionality of TCP handles such tasks as the detection of data loss, coordination for the retransmission of packets, re-ordering packets that arrive out-of-order, and governing the rate at which packets are transmitted over the connection [15]. In doing so, TCP provides reliable data transfer by ensuring that packets arrive at the intended recipient.



C E U A P R S F

Congestion Window
C 0x80 Reduced (CWR)
E 0x40 ECN Echo (ECE)
U 0x20 Urgent
A 0x10 Ack
P 0x08 Push
R 0x04 Reset
S 0x02 Syn
F 0x01 Fin

ECN (Explicit Congestion Notification). See RFC 3168 for full details, valid states below.

Packet State	DSB	ECN bits
Syn	0 0	1 1
Syn-Ack	0 0	0 1
Ack	0 1	0 0
No Congestion	0 1	0 0
No Congestion	1 0	0 0
Congestion	1 1	0 0
Receiver Response	1 1	0 1
Sender Response	1 1	1 1

0 End of Options List
1 No Operation (NOP, Pad)
2 Maximum segment size
3 Window Scale
4 Selective ACK ok
8 Timestamp

Checksum of entire TCP segment and pseudo header (parts of IP header)

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

Please refer to RFC 793 for the complete Transmission Control Protocol (TCP) Specification.

Figure 9. TCP Packet Header Data Fields. Source: [13].

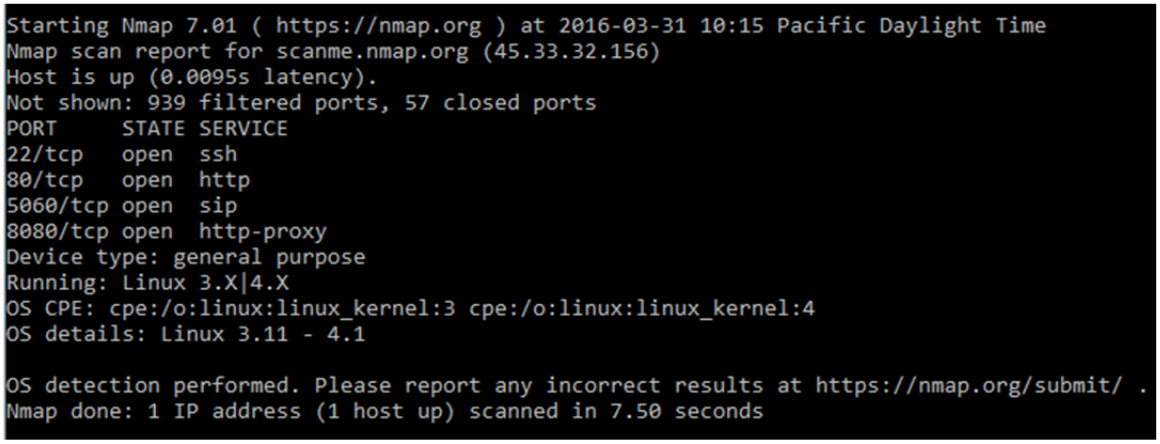
The functionality of TCP allows for the decoupling of the protocols that operate at the application layer from the specific details of actually reliably exchanging data over the network. Additionally, TCP's use of stream sockets to establish these host-to-host connections allows for multiple connections to be active on the same device at once.

C. EXISTING TOOLS AND RESOURCES

It is clear that there is an abundance of information available in the data packets flowing over a network. The process of extracting this data into meaningful results is tedious and must be automated. As discussed by McBride in chapter II, section C, of his thesis [1], there are many publicly available tools that allow network administrators to monitor network traffic and conduct manual analysis. The functionality of these tools can be leveraged in an all-inclusive program that will allow a user to automatically scan and verify a network. Representing the network itself in a succinct manner presents a different challenge. A generally accepted practice is to represent a network as a graph: a series of vertices and connecting edges. The vertices represent devices on the network and the edges represent connections between individual devices. Fortunately, there exist programming libraries and markup languages that provide functionality for the storage, manipulation, and visualization of a network graph. This thesis focuses on three existing tools that provide functionality to scan and analyze network traffic, visualize and manipulate the data available, and store the data for later use by other modules within the MAVNATT system.

1. NMap Security Scanner

The NMap Security Scanner is a comprehensive open-source tool that provides functionality for active network traffic analysis and security auditing [13]. It is available as a command-line tool or in a graphical user interface for Linux, MAC OS X, and Windows operating systems. NMap boasts an extensive service set with more than 100 command-line options that include host discovery, remote host operating system (OS) detection, port scanning, service and application version detection, and firewall detection [13]. The tool provides multiple options for output of results, which can be parsed easily or formatted into a human-readable form (see Figure 10).



```
Starting Nmap 7.01 ( https://nmap.org ) at 2016-03-31 10:15 Pacific Daylight Time
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.0095s latency).
Not shown: 939 filtered ports, 57 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
5060/tcp open  sip
8080/tcp open  http-proxy
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.11 - 4.1

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.50 seconds
```

Notable data includes name to IP address resolution, significant open ports, and candidate OS based on fingerprinting techniques.

Figure 10. Example Output of an NMap Scan on a Remote Host.

While a complete understanding of all available NMap operations requires extensive experience with the tool, the ability to quickly and accurately scan a network for mapping purposes can be achieved with a few basic commands [13]:

- **NMap default functionality:** Scans can be conducted on individual hosts, IP ranges, or entire subnets using CIDR notation, as exemplified below, respectively. NMAP checks to see if the host is up, then conducts a scan on ports 1-1000 to determine which ports of those are open [13].

  ```
  nmap 192.168.1.1,192.168.1.2
  nmap 192.168.1.1-30
  nmap 192.168.1.0/24
  ```

- **Host discovery (-sn):** Omit the port scan and only return those hosts that are detected [13].

  ```
  nmap –sn 192.168.1.0/24
  ```

- **Remote OS detection (-O):** NMap maintains an internal database of fingerprints based on how different systems respond to TCP/IP probes. This tool compares received results to determine a potential OS match [13].

  ```
  nmap –O 192.168.1.0/24
  ```

- **ARP Scan (-PR):** Sends an ARP Request message vice ICMP or other IP based probe. This facilitates quicker scanning on LANs, especially those that are sparsely populated with hosts. NMap automatically recognizes if a host address is on the same subnet as the local host from which the scan is conducted and defaults to the ARP scan [13].

```
nmap –PR 192.168.1.0/24
```

- **Exclude addresses (--exclude):** Scan all hosts on the given subnet or address list except those listed in the exclusion list [13].

```
nmap 192.168.1.0/24 --exclude 192.168.1.1
```

- **Traceroute (--traceroute):** Displays additional information about the route from the local machine to the remote host, including the IP address of each intermediate hop and timing information [13].

```
nmap 192.168.1.1 --traceroute
```

These standard commands can be combined or used individually to retrieve enough information necessary to derive the topology layout of a LAN along with configurations and details of devices on the network.

2. Java Universal Network/Graph Framework

The Java Universal Network/Graph (JUNG) Framework is a collection of open-source Java libraries that provide functionality for manipulating information represented as a network graph. Created by three PhD students at the University of California, Irvine, the project has been continually revised and updated since 2003 [16]. The nature of computer networks makes JUNG an excellent tool for visualizing the network topology. Using the visualization library, we can display a graphical representation of the vertices and edges in a network (see Figure 11). Features of the library include functionality for labeling graph elements, changing the color, shape, and size of vertices and edges, and manipulating the position of graph elements via mouse [17]. Additionally, JUNG includes several pre-defined algorithms that automatically determine the position at which to draw each graph element, providing a clean display for easy viewing.

The underlying infrastructure of JUNG permits full customization of the data used to represent vertices and edges in a graph. This allows for the creation of custom objects that can be used to store all necessary information about network devices such as routers, switches and clients. These custom objects can then be set as the graph vertices, with connection information stored in custom graph edge objects.

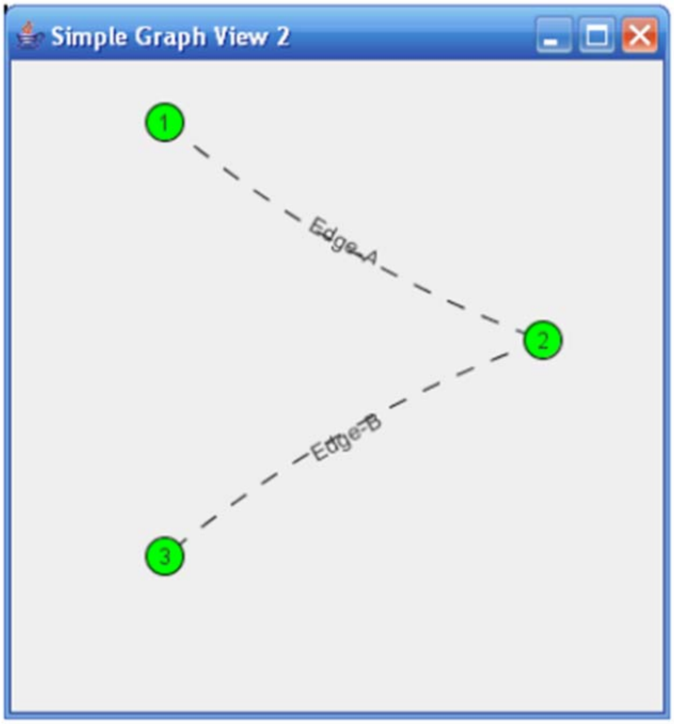
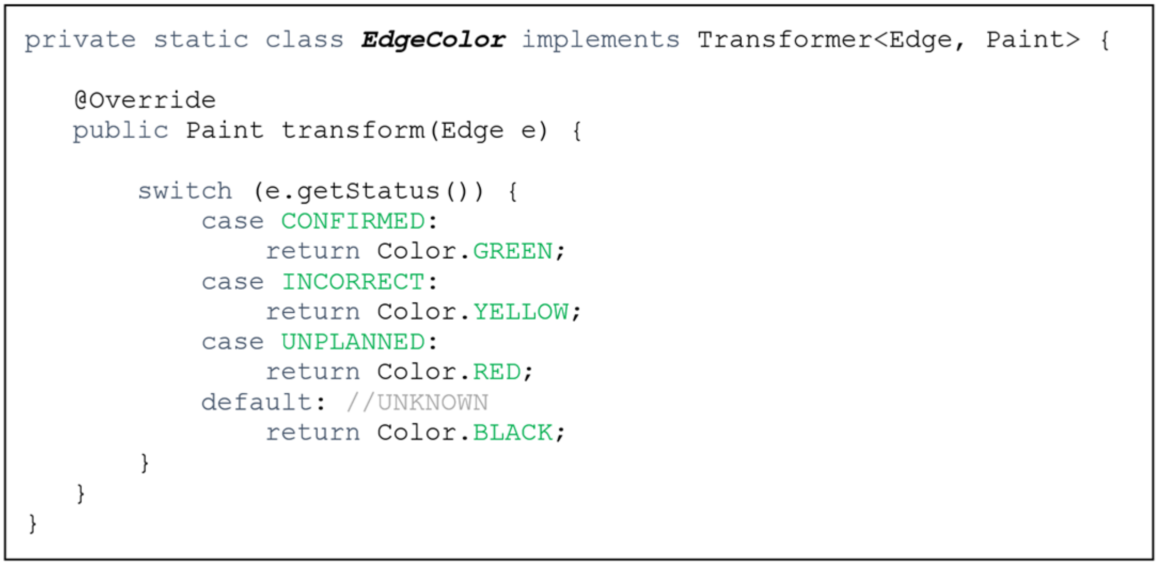


Figure 11. Sample Graph Display Using the JUNG Visualization Library. Source: [17].

A key component of the JUNG functionality is the Transformer interface, which allows the user to set or retrieve information stored within the vertex and edge objects during runtime. This is done by defining a class implementation of the Transformer interface for each necessary task [17]. The defined class conducts a mapping from one object to another. In this manner, the Transformer can change the display characteristics of the object based on the values of each vertex or edge. The code snippet shown in Figure 12 is an example Transformer implementation that dynamically colors a graph edge based on the status of the object used to represent that edge.



```
private static class EdgeColor implements Transformer<Edge, Paint> {

   @Override
   public Paint transform(Edge e) {

       switch (e.getStatus()) {
           case CONFIRMED:
               return Color.GREEN;
           case INCORRECT:
               return Color.YELLOW;
           case UNPLANNED:
               return Color.RED;
           default: //UNKNOWN
               return Color.BLACK;
       }
   }
}
```

This Transformer class maps a Paint object (Color) to an Edge object based on the status of the Edge.

Figure 12. Example JUNG Transformer Implementation.

3. GraphML File Format

The GraphML file format is a markup language designed to efficiently represent the topology of a network graph, allowing data to be saved for later use via import and export operations [18]. Derived from the Extensible Markup Language (XML) [19], GraphML uses start and end tags to declare the different components of a graph. Empty tags may be used when declaring an element that has no content. Attributes for the individual components can be defined within the start tag of the component. Additionally, GraphML allows for the storage of data on nodes and edges by defining custom attributes of primitive data types. Key components of a GraphML file are:

- **Header:** XML version and schema information used to interpret and validate the file format. Schema details are listed inside the GraphML start tag. The remainder of declarations of the file are nested in the content section between the GraphML start and end tags [18].

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns=http://graphml.graphdrawing.org/xmlns>
...
</graphml>
```

- **Graph:** The graph itself. An edge default attribute of directed, undirected, or mixed must be defined [18].

  ```
  <graph id="Simple Graph" edgedefault="undirected">
  ...
  </graph>
  ```

- **Node:** A vertex in the graph. A unique identifier must be defined for each node. Node declarations are nested in the graph content section [18].

  ```
  <node id="n1"/>
  <node id="n2"/>
  ```

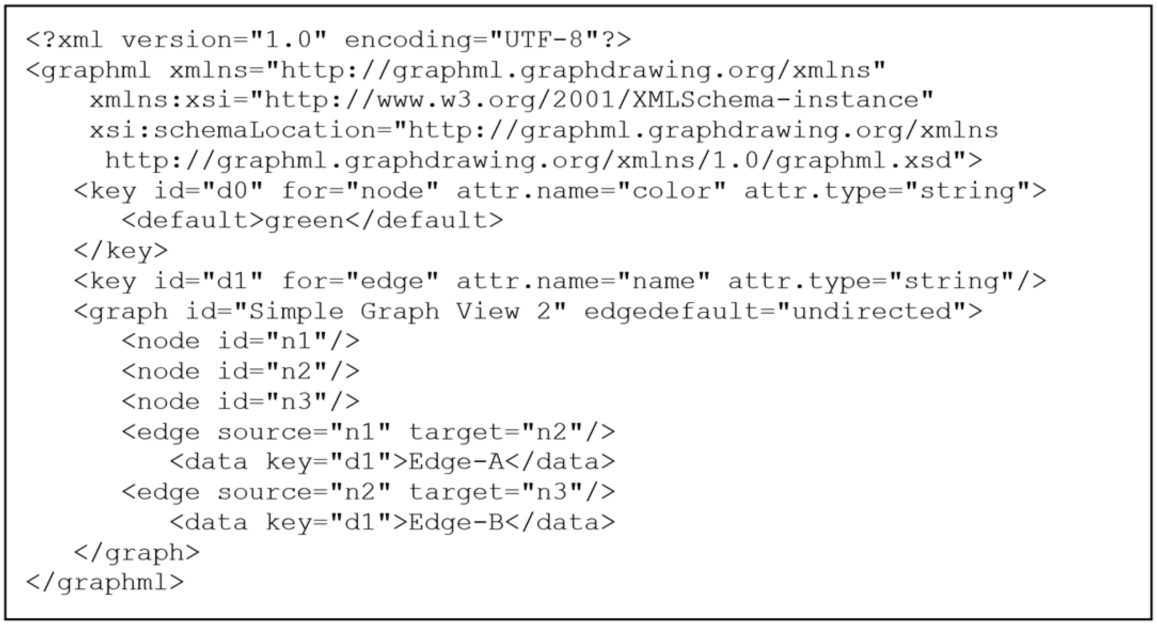
- **Edge:** An edge between two nodes in the graph. Source and target nodes must be defined for each edge. Edge declarations are nested in the graph content section [18].

  ```
  <edge source="n1" target="n2"/>
  ```

- **Custom Attributes:** Custom attributes are declared outside of the graph element using the key tag, while values for those attributes can be set for individual components within their respective content sections using the data tag. Custom attributes have their own attributes of identifier, name, type, and domain that are defined inside the start tag of the attribute declaration. Finally, a default value can be declared for custom attributes [18].

  ```
  <key id="d0" for="node" attr.name="x" attr.type="int">
  <key id="d1" for="node" attr.name="y" attr.type="int">
  ...
  <node id="n1">
       <data key="d0">50</data>
       <data key="d1">100</data>
  </node>
  ```

The network graph shown in Figure 11 can be represented in a very simple manner by declaring custom attributes and setting their appropriate values for components in the network (see Figure 13). While networks can be succinctly defined using the GraphML file format, there is no default application for reading and displaying a graph defined in GraphML. Implementation is left to the user and can be accomplished using open source libraries such as JUNG or other XML parsing methods.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
     http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
   <key id="d0" for="node" attr.name="color" attr.type="string">
      <default>green</default>
   </key>
   <key id="d1" for="edge" attr.name="name" attr.type="string"/>
   <graph id="Simple Graph View 2" edgedefault="undirected">
      <node id="n1"/>
      <node id="n2"/>
      <node id="n3"/>
      <edge source="n1" target="n2"/>
         <data key="d1">Edge-A</data>
      <edge source="n2" target="n3"/>
         <data key="d1">Edge-B</data>
   </graph>
</graphml>
```

Figure 13. GraphML Representation of the Network Graph Shown in Figure 11.

D. SUMMARY

This chapter discussed the format, availability, and flow of data on a LAN, as well as some proven methodologies for extracting and using that data to represent the topology of the network. Fundamental objectives of a topology mapping application are algorithmic analysis of the available data, accurate representation of the network, and a user-friendly interface for visualization and verification of the network plan. The resources described in this chapter can be leveraged to compose an automated application for scanning and visualizing network topology.

III. DESIGN AND IMPLEMENTATION

A. APPLICATION DESCRIPTION

The goal of this thesis is to develop an application for use by network professionals with administrative oversight over LANs such as those employed at the tactical or small unit level. Key functional points of the application would allow the user to import a proposed network plan, verify the plan, scan the live network based on expected values, and output results in a succinct and readable manner. This would allow the network administrator to accurately represent a network, both planned and actual, in order to produce a virtualized replica of the LAN for use in training scenarios. This application represents the mapping portion of the proposed MAVNATT system [1].

The mapping application was written in Java Standard Edition version 8 using the Java Development Kit with Netbeans 8.1 Integrated Development Environment [20]. The program employs JUNG libraries [16] to manipulate and visualize the network, and NMap [13] to scan the live network. Output of the program can be saved as a human-readable text file or as a GraphML [18] file for import and export operations. The application was written on a Windows 10 platform and compiled with all external libraries and dependencies. This allows the application to run on any operating system, provided that Java and NMap are installed on that platform.

B. OBJECT-ORIENTED FRAMEWORK

We chose to take an object-oriented programming approach for application development. This allowed us to create a framework to represent a network via custom data structures which fit our needs of representing specific data points within the network, as well as the complicated relational interactions between network entities. Additionally, an object-oriented approach allowed us to fully leverage the portability provided by the programming environment, removing dependencies on platform-specific native data sets. In this thesis, we describe each custom object using the Unified Modeling Language (UML) [21]. For brevity, the UML diagrams in this section only include key data fields and methods within each object, omitting common methods such as toString() methods and attribute getters and setters. The complete UML diagram for the network framework can be seen in Appendix A.

1. Interface and SwitchInterface Objects

The Interface object (Figure 14) represents the foundational component of a network. Individual devices connect to a live network using a network interface card (NIC) and point-to-point connections are from one NIC to another. The Interface fields are used to characterize the object and its status on the network. Two Interface objects are considered to be equal if they share either the same MAC address or the same IPv4 address.

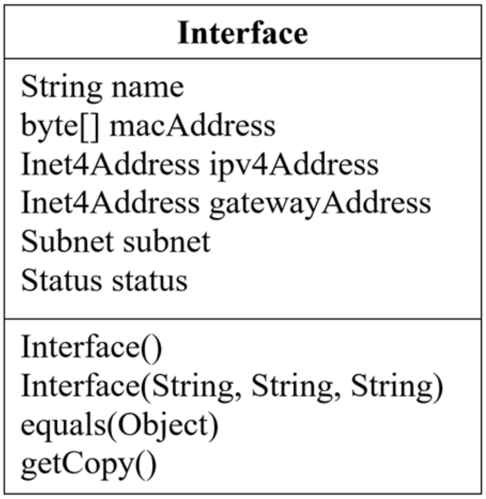


Figure 14. UML Diagram for Interface Object.

The SwitchInterface object (Figure 15) extends the Interface object and is used to represent a port on a layer 2 switch. The key distinction is that a layer 2 switch port is transparent on the network, without an IPv4 address or MAC address. We identify a SwitchInterface by the name of the switch to which it belongs and a unique numeric identifier. Two SwitchInterface objects are considered equal if they have the same switch name and unique identifier.

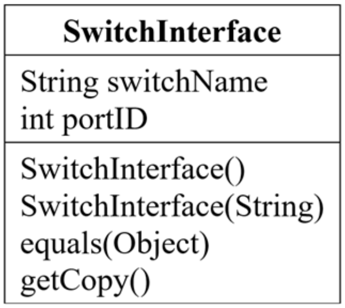


Figure 15. UML Diagram for SwitchInterface Object.

2. Node Object

The Node object (Figure 16) is used to represent any physical device on the network: router, switch, hub, computer, printer, etc. Each Node maintains a set of Interface objects, allowing correct representation of multiple network connections for a single device (e.g., routers). The merge() method can be used to combine two Node objects. Two Node objects are considered equal if they have the same name or share at least one Interface.

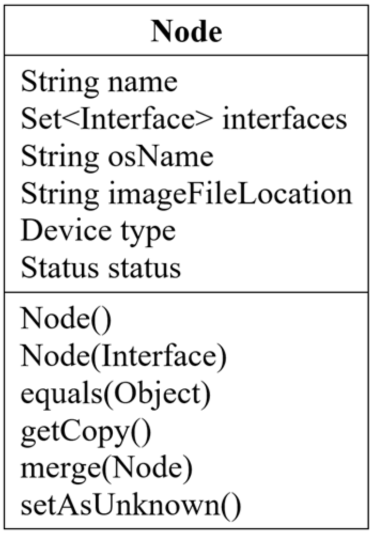


Figure 16. UML Diagram for Node Object.

3. Edge Object

The Edge object (Figure 17) is used to represent a connection between two Interfaces. Two Edge objects are considered equal if both represent the same two Interface endpoints.

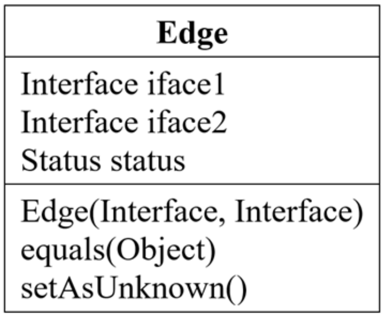


Figure 17. UML Diagram for Edge Object.

4. Subnet Object

The Subnet object (Figure 18) is used to represent a subnet address. It provides functionality to determine whether a given IPv4 address is a part of the subnet. Two Subnet objects are considered equal if they have the same subnet address and subnet mask. The netIDsEqual() method can be used to determine whether two different subnets have the same subnet address, indicating a potential error in the subnet allocation scheme.

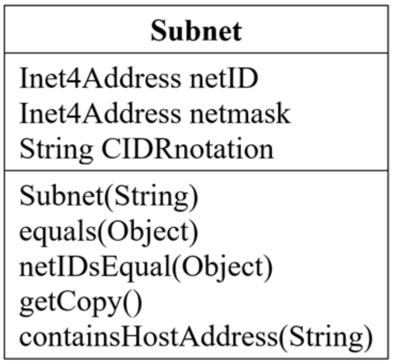


Figure 18. UML Diagram for Subnet Object.

5. Status and Device Enumerations

The Status Enumeration (Figure 19) is used throughout the framework as a flagging mechanism for Interfaces, Edges, and Nodes. These flags are used when scanning the live network and comparing found results with expected results. Status is also used to determine which color to make the entity during visualization. Definitions of the individual Status flags are:

- **PLANNED (blue):** This entity is represented on the network plan.
- **CONFIRMED (green):** This entity was confirmed during the live network scan with details matching those represented on the network plan.
- **MISSING (orange):** This entity is on the network plan, but was not found during the scan of the live network.
- **INCORRECT (yellow):** This entity was found during the live network scan, but some data discovered for the actual device conflicts with that represented on the network plan.
- **UNPLANNED (red):** This entity is not represented on the network plan, but was discovered during a scan of the live network.
- **UNKNOWN (brown):** This entity is represented on the network plan, but there was not enough information available during the live network scan to make a status determination.

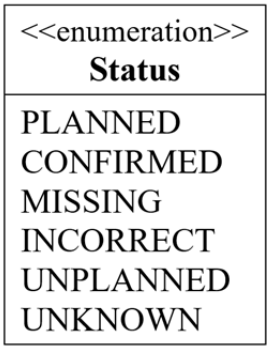


Figure 19. UML Diagram for Status Enumeration.

The Device Enumeration (Figure 20) is used to specify the type of physical device represented by a Node object. This information is used when visualizing the network and when generating a GraphML file for later virtualization. The format of the Device enumeration facilitates easy expansion or modification of device types in future implementations.

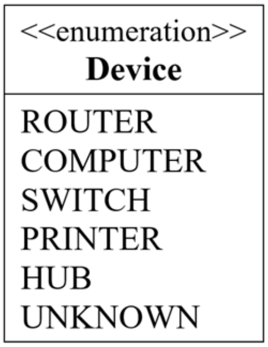


Figure 20. UML Diagram for Device Enumeration.

6. Network Object

The Network object (Figure 21) is used to bring all of the objects in the framework together. Individual fields track the Nodes, Edges, and Subnets that comprise the Network, as well as a mapping from Interfaces to Nodes. Individual methods provide functionality for adding Nodes and Edges to the Network, merging two Networks together, retrieving specific Nodes and Interfaces based on defining details, and converting the Network to a graph for use by the JUNG libraries. Two Network objects are considered equal only if they have the same set of Nodes, Edges, and Node-to-Interface mapping.

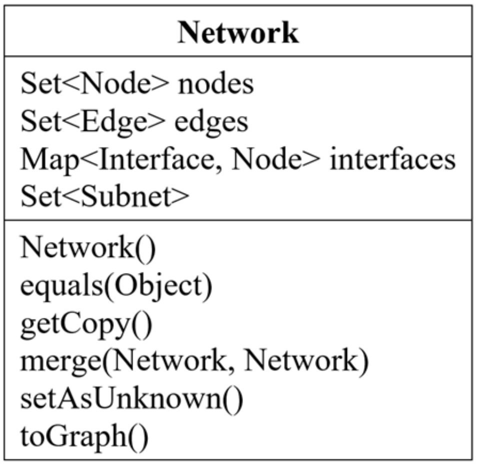


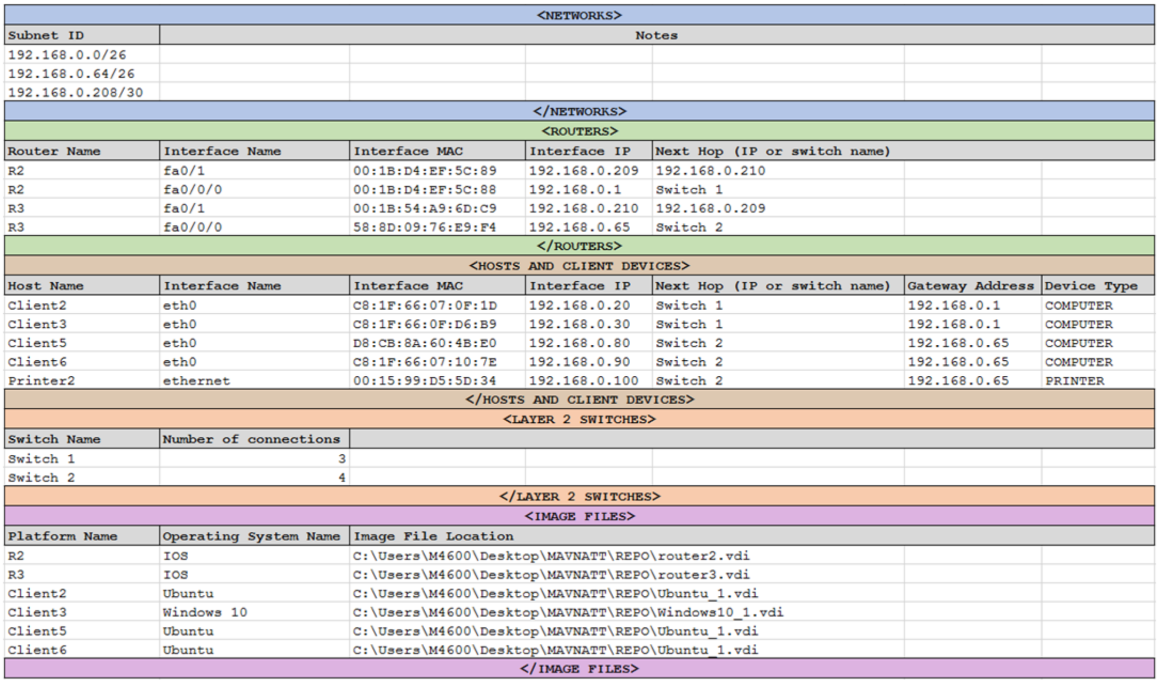
Figure 21. UML Diagram for Network Object.

C. PROGRAM FLOW OF EXECUTION

Using the framework described in Section B above, we developed an application that algorithmically scans a live network and provides results by comparing the discovered topology to that of the planned network. The flow of execution for the program starts with manual generation of a network plan, visualization of that plan, a live network scan with visualization of results, and functionality to store and print results in different formats.

1. Network Plan Generation

The very first step in the establishment of any computer network is the development of a network plan. This is the necessary starting point and requires manual planning by the network administrator to determine defining characteristics of the network such as overall layout and topology, details of connected devices, logical address allocation, and security controls in place. While developing a format for the network plan, our primary goal was to represent all necessary entities in a succinct but complete manner while still being easy to use and read. A secondary goal was to make the plan as simple as possible, without reliance on complex database structures or external tracking systems, as the network plan would need to be flattened to a text file to facilitate import and parsing into our application. The resulting network plan format shown in Figure 22 uses XML-style tags to separate sections of the plan, each with information relevant to that network entity. The plan is generated in Microsoft Excel and saved as an .xlsx file type. This allows for the creation of a succinct yet detailed plan that can be exported to a text file for later parsing.



<NETWORKS>						
Subnet ID	Notes					
192.168.0.0/26						
192.168.0.64/26						
192.168.0.208/30						
</NETWORKS>						
<ROUTERS>						
Router Name	Interface Name	Interface MAC	Interface IP	Next Hop (IP or switch name)		
R2	fa0/1	00:1B:D4:EF:5C:89	192.168.0.209	192.168.0.210		
R2	fa0/0/0	00:1B:D4:EF:5C:88	192.168.0.1	Switch 1		
R3	fa0/1	00:1B:54:A9:6D:C9	192.168.0.210	192.168.0.209		
R3	fa0/0/0	58:8D:09:76:E9:F4	192.168.0.65	Switch 2		
</ROUTERS>						
<HOSTS AND CLIENT DEVICES>						
Host Name	Interface Name	Interface MAC	Interface IP	Next Hop (IP or switch name)	Gateway Address	Device Type
Client2	eth0	C8:1F:66:07:0F:1D	192.168.0.20	Switch 1	192.168.0.1	COMPUTER
Client3	eth0	C8:1F:66:0F:D6:B9	192.168.0.30	Switch 1	192.168.0.1	COMPUTER
Client5	eth0	D8:CB:8A:60:4B:E0	192.168.0.80	Switch 2	192.168.0.65	COMPUTER
Client6	eth0	C8:1F:66:07:10:7E	192.168.0.90	Switch 2	192.168.0.65	COMPUTER
Printer2	ethernet	00:15:99:D5:5D:34	192.168.0.100	Switch 2	192.168.0.65	PRINTER
</HOSTS AND CLIENT DEVICES>						
<LAYER 2 SWITCHES>						
Switch Name	Number of connections					
Switch 1	3					
Switch 2	4					
</LAYER 2 SWITCHES>						
<IMAGE FILES>						
Platform Name	Operating System Name	Image File Location				
R2	IOS	C:\Users\M4600\Desktop\MAVNATT\REPO\router2.vdi				
R3	IOS	C:\Users\M4600\Desktop\MAVNATT\REPO\router3.vdi				
Client2	Ubuntu	C:\Users\M4600\Desktop\MAVNATT\REPO\Ubuntu_1.vdi				
Client3	Windows 10	C:\Users\M4600\Desktop\MAVNATT\REPO\Windows10_1.vdi				
Client5	Ubuntu	C:\Users\M4600\Desktop\MAVNATT\REPO\Ubuntu_1.vdi				
Client6	Ubuntu	C:\Users\M4600\Desktop\MAVNATT\REPO\Ubuntu_1.vdi				
</IMAGE FILES>						

Figure 22. Example Network Plan Format.

Clarification of the network plan sections:

- **<NETWORKS>:** This section is used to list all subnets that will comprise the LAN in CIDR notation, one per row.

- **<ROUTERS>:** This section is used to list all routers on the LAN. Each interface on a router must be declared on a separate row, allowing for multiple interfaces on the same device. The "Next Hop" column is used to identify a physical connection within the network and should represent the address of the interface to which this interface is physically connected. The name of any switch devices listed as next hop connections must exactly match a switch name listed in the <LAYER 2 SWITCHES> section.

- **<HOSTS AND CLIENT DEVICES>:** Similar to the <ROUTERS> section, this section lists all client devices on the network, one interface per row. The "Device Type" column is a dropdown menu that provides possible options. This field is used to select an icon for the device during visualization.

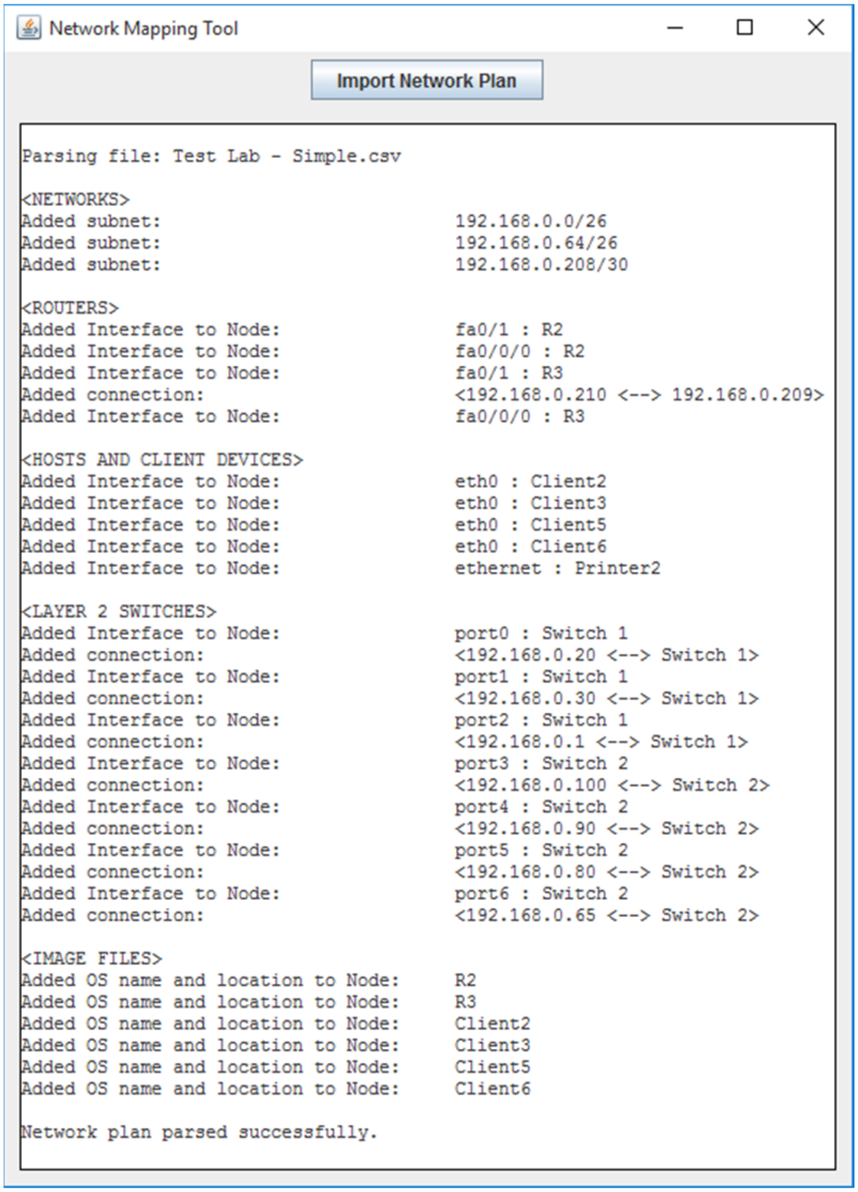
- **<LAYER 2 SWITCHES>:** Any layer 2 switch with connections on the network must be listed in this section, one per row. Switch names listed in the "Next Hop" column of the previous two sections must exactly match at least one switch name listed in this section.

- **<IMAGE FILES>:** Primarily for future use when integrating the MAVNATT mapping module with the virtualization module, this section allows the network administrator to list the OS type and the location of virtual machine image files for specific network entities. This data is stored on the Node object and can be used to instantiate virtual machines of the given type.

After completion of the network plan, it can be exported as a comma-separated values (CSV) file type. This provides the file flattening required to import and parse the plan for use in the mapping application. This network plan serves as the expected value against which the live network scan results can be compared. Appendix B shows a CSV version of the network plan shown in Figure 22.

2. Network Plan Import and Visualization

The network plan is created manually, independent of the mapping application. The exported CSV file then serves as the starting point for the application, with file import being the only option upon startup, via the "Import Network Plan" button. After import, the console displays a log (see Figure 23), which reflects progress as the plan is parsed and individual objects are created within the network framework as described in Chapter III, Section B. Potential problems identified while parsing the network plan are displayed in the log as well. This provides the network administrator an opportunity to identify mistakes in the plan.



```
Parsing file: Test Lab - Simple.csv

<NETWORKS>
Added subnet:                           192.168.0.0/26
Added subnet:                           192.168.0.64/26
Added subnet:                           192.168.0.208/30

<ROUTERS>
Added Interface to Node:                fa0/1 : R2
Added Interface to Node:                fa0/0/0 : R2
Added Interface to Node:                fa0/1 : R3
Added connection:                       <192.168.0.210 <--> 192.168.0.209>
Added Interface to Node:                fa0/0/0 : R3

<HOSTS AND CLIENT DEVICES>
Added Interface to Node:                eth0 : Client2
Added Interface to Node:                eth0 : Client3
Added Interface to Node:                eth0 : Client5
Added Interface to Node:                eth0 : Client6
Added Interface to Node:                ethernet : Printer2

<LAYER 2 SWITCHES>
Added Interface to Node:                port0 : Switch 1
Added connection:                       <192.168.0.20 <--> Switch 1>
Added Interface to Node:                port1 : Switch 1
Added connection:                       <192.168.0.30 <--> Switch 1>
Added Interface to Node:                port2 : Switch 1
Added connection:                       <192.168.0.1 <--> Switch 1>
Added Interface to Node:                port3 : Switch 2
Added connection:                       <192.168.0.100 <--> Switch 2>
Added Interface to Node:                port4 : Switch 2
Added connection:                       <192.168.0.90 <--> Switch 2>
Added Interface to Node:                port5 : Switch 2
Added connection:                       <192.168.0.80 <--> Switch 2>
Added Interface to Node:                port6 : Switch 2
Added connection:                       <192.168.0.65 <--> Switch 2>

<IMAGE FILES>
Added OS name and location to Node:     R2
Added OS name and location to Node:     R3
Added OS name and location to Node:     Client2
Added OS name and location to Node:     Client3
Added OS name and location to Node:     Client5
Added OS name and location to Node:     Client6

Network plan parsed successfully.
```

The application parses each section of the network plan sequentially and displays a progress log. Updates are displayed as objects are created within the network framework, or when potential problems are identified.

Figure 23. Import Log After Parsing the Network Plan Shown in Figure 22.

Upon successful completion of the network plan import, the application automatically displays a visualization of the planned network (see Figure 24). This provides the network administrator another tool with which he can verify the plan and identify any potential problems in the proposed configuration. The network is displayed as a graph using the JUNG libraries [16]. Each device is represented by a corresponding icon, with edges of the graph representing physical connections between devices. The blue icons in the display indicate that this is the planned network and the status variable of each Node, Edge, and Interface object is set to "PLANNED."

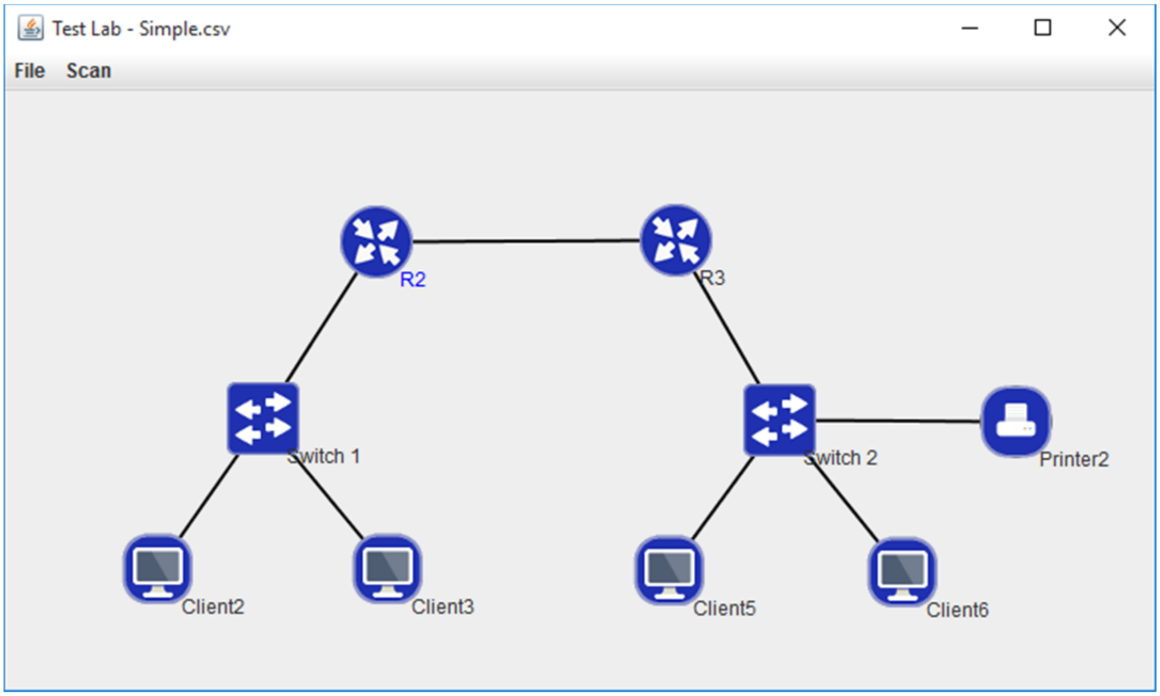


Figure 24. Visualization after Parsing the Network Plan Shown in Figure 22.

3. Live Network Scan and Visualization

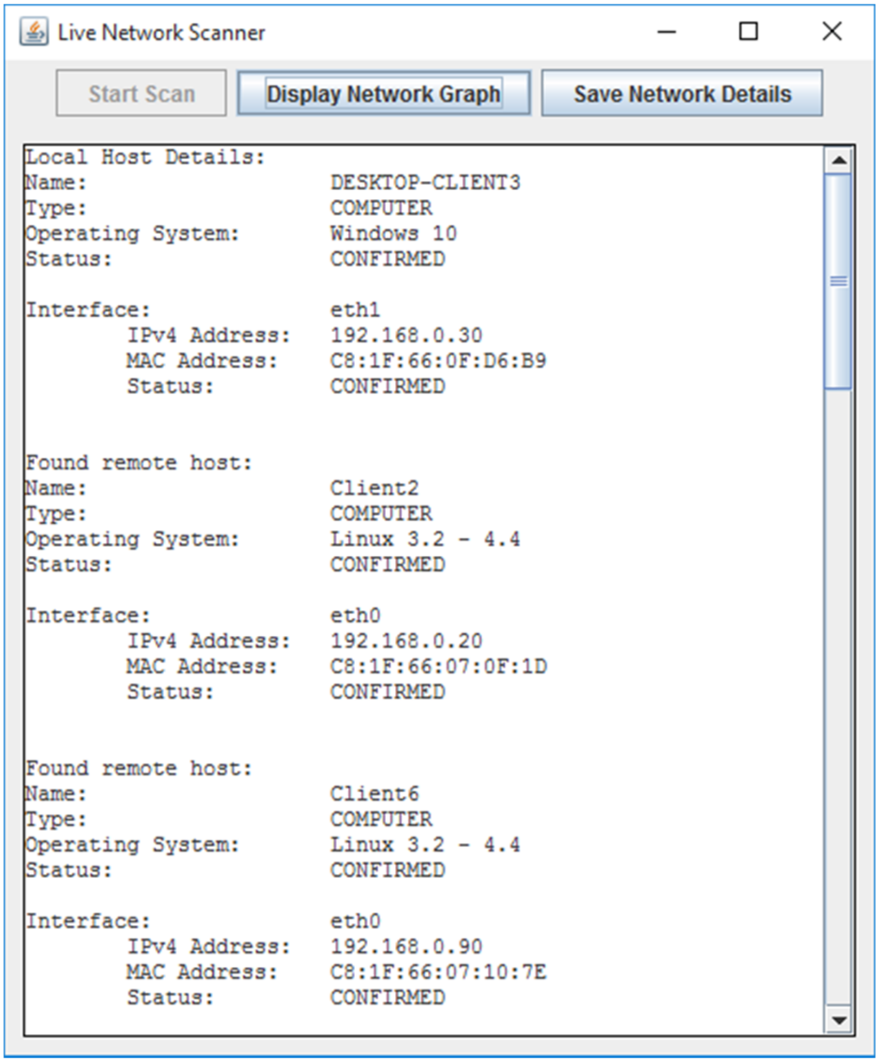
At this point of the application execution phase, the planned network is stored in memory as a Network object according to the object-oriented framework described in Chapter III, Section B. Here, the user can scan the live network to detect configuration and settings of connected devices. The live network scan is a sequential process using bounds established by the planned network, which serves as the expected value against which results are compared. Any variation between the planned network and discovered results is annotated and displayed. The scan is initiated via the menu bar in the graph visualization window by selecting the Scan > Scan Live Network option. This displays a new empty console with one available option, the "Start Scan" button.

a. Preparation

The application starts by making a deep copy of the planned network. During this process, the status variable of each Node, Edge, and Interface object is set to "UNKNOWN." This process is transparent to the user and facilitates status updates on individual network entities as settings are confirmed or found to be invalid. The planned network is left intact and this new Network object, referred to as the result network, is now the working copy for the live network scan.

b. Detect Local Host Settings

In the initial verification step, the application determines the settings for the local host on which the mapping application is being executed. Once the local IP address and MAC address are determined, the application searches the result network for the corresponding Interface object with matching details. Details of the local host are displayed to the scan report console (see Figure 25) along with any inconsistencies discovered. This identification of the local host provides the application with necessary information to conduct a more fine-grained verification of other devices on the same subnet, as this is the only scenario when the MAC address of remote devices is visible to the mapping application. The status variable for the discovered Interface is set accordingly.



```
Local Host Details:
Name:                   DESKTOP-CLIENT3
Type:                   COMPUTER
Operating System:       Windows 10
Status:                 CONFIRMED

Interface:              eth1
        IPv4 Address:   192.168.0.30
        MAC Address:    C8:1F:66:0F:D6:B9
        Status:         CONFIRMED


Found remote host:
Name:                   Client2
Type:                   COMPUTER
Operating System:       Linux 3.2 - 4.4
Status:                 CONFIRMED

Interface:              eth0
        IPv4 Address:   192.168.0.20
        MAC Address:    C8:1F:66:07:0F:1D
        Status:         CONFIRMED


Found remote host:
Name:                   Client6
Type:                   COMPUTER
Operating System:       Linux 3.2 - 4.4
Status:                 CONFIRMED

Interface:              eth0
        IPv4 Address:   192.168.0.90
        MAC Address:    C8:1F:66:07:10:7E
        Status:         CONFIRMED
```

Figure 25. Example Scan Report Console Output.

c. *Scan Planned Devices*

At this stage, the application iterates through each Interface object in the result network. For those Interfaces with an IP address defined (i.e., not SwitchInterface objects), the application conducts the following NMap scan, where [IP ADDRESS] is the IP address of the current Interface being verified.

```
nmap -F -O --osscan-limit --max-os-tries 1 [IP ADDRESS]
```

This NMap scan is executed in a background process and the output is redirected to the mapping application where it is parsed. If a host is detected at the given IP address, a parsing function uses regular expressions to extract the IP address, MAC address (if present), and potential OS of the host from the NMAP scan report. This data is used to verify the details of the expected Interface object and the status variable for the Interface is set accordingly. As data is parsed, log messages are displayed to the scan report console, as shown in Figure 25, and changes to each Interface in the result network are made according to the following criteria:

- Interfaces on the same subnet as the local host are labeled "CONFIRMED" if the discovered IP address and MAC address match expected values.
- Interfaces on the same subnet as the local host are labeled "INCORRECT" if the discovered MAC address differs from the expected value.
- Interfaces on remote subnets are marked "CONFIRMED" if their IP address alone matches its expected value.
- Any status change made to an Interface is also made to the Node object to which that Interface belongs.
- If the NMap scan returns a single potential OS match for the scanned host, the Node object to which the Interface belongs is automatically updated to reflect the discovered value.
- If no corresponding host is found via the NMap scan, no action is taken at this time.

After scanning for all Interface objects, the mapping application will rescan for all Interfaces in the result network that still have a status of "UNKNOWN," meaning they were not detected on the first attempt. If NMap again fails to locate a host at the given IP address, the status variable for the given Interface is set to "MISSING." The status of any Node object with all Interfaces set as "MISSING" is set to "MISSING" as well. If a Node has at least one missing Interface and at least one that was detected during the NMap scan, the status for that Node is set to "INCORRECT."

d. Scan for Unplanned Devices

After scanning for all expected devices, the mapping application scans the known subnets for any device present that is not on the network plan. To do so, the application executes the following NMap scan, where [SUBNETS] is a list of all subnets from the planned network and [ADDRESSES] is a list of all IP addresses of Interfaces from the planned network.

```
nmap –sn [SUBNETS] --exclude [ADDRESSES]
```

If an unexpected device is found, the NMap results are parsed for IP address and MAC address (if present) using regular expressions. These addresses are used to create new Interface and Node objects with the status variable set to “UNPLANNED.” The new Node with corresponding Interface are then added to the result network, allowing them to be displayed later.

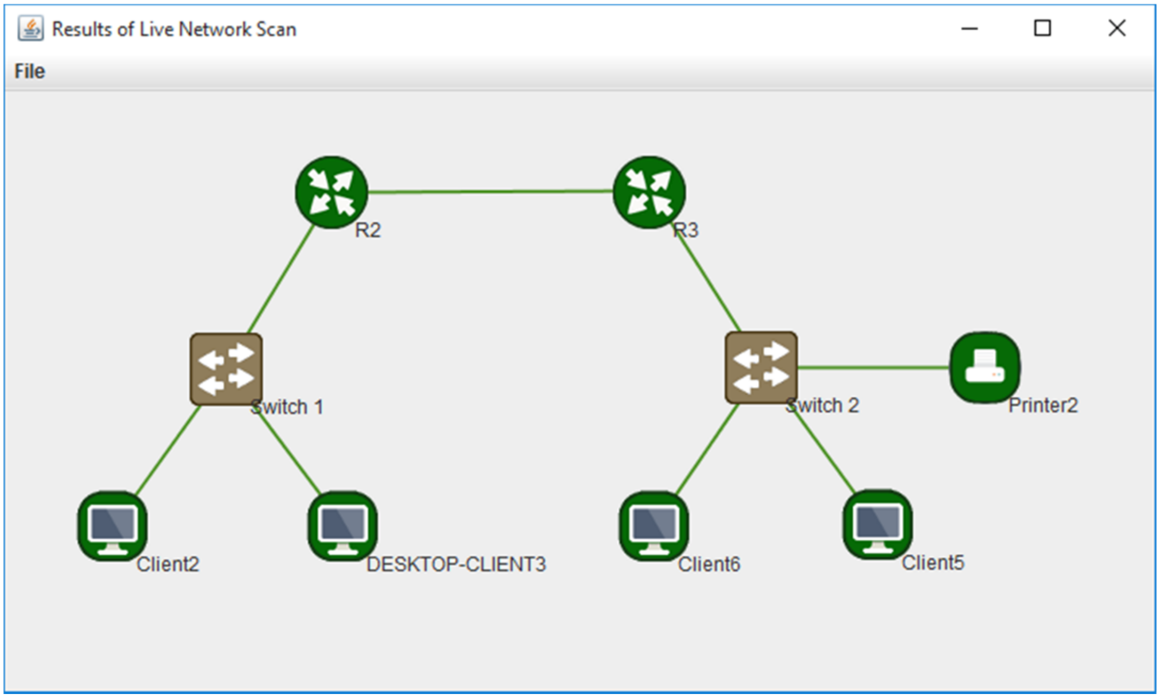
e. Analyze Network Connections

Checking the status of the connections in the network to complete the comparison between the live network scan results and the expected value required that we make some inferences. The layer 2 switch is transparent on the network, making it impossible to scan for the existence of specific switch ports. Instead, the application counts the number of discovered IP addresses on each subnet. If there are more than two addresses on a given subnet, then all connections must be going through a switch or hub. Changes to each Edge in the result network are made according to the following criteria:

- If one endpoint is an Interface with an IP address on a subnet with more than two IP addresses present, the other endpoint must be a SwitchInterface. If the Edge meets this criteria, its status is set to “CONFIRMED.” Otherwise, it is set to “INCORRECT.”
- If one endpoint Interface is marked as “MISSING,” the Edge status is set to “MISSING.”
- If both endpoints are Interfaces with IP addresses on the same subnet, and the number of IP addresses present is exactly two, the Edge status is set to “CONFIRMED.”
- At no time is any new Edge object created and added to the result network.

4. Results Output

Upon completion of the network scan, the application presents several methods to view and save the results. As shown in Figure 25, the user is given options to display the network graph or save the network details. The "Display Network Graph" button shows a visualization of the live network scan results (see Figure 26). Nodes and Edges in the display are color coded according the status variable for the given object. This provides the network administrator with an overview of the network configuration that allows for quick identification of potential problems.



In this example, all interfaces on all devices have been confirmed by the network scan. All connections meet the confirmation criteria. The layer 2 switches maintain a status of UNKNOWN.

Figure 26. Visualization of Live Network Scan Results.

The "Save Network Details" option allows the user to save a textual representation of the live network scan results. Appendix C is the detailed log file for the live network scan shown in Figure 26.

Finally, the user has the option to save a graph representation of a network, either planned or actual scan results, as a GraphML file. This is accomplished via the File > Save as GraphML option from any network visualization window.

D. SUMMARY

This chapter details the internal structure of the MAVNATT mapping application as well as the program flow of execution. The established data structures in the network framework allow for extensibility and future development. Additional data fields can be added to individual object types without disrupting the relational interaction between each object within the framework, allowing for finer granularity of details of connected devices. The program flow of execution allows the network administrator to visualize a proposed network plan and quickly identify potential problems. Similarly, the available output options of a live network scan allow the user to easily identify conflicts between the planned network and actual results. This functionality facilitates network configuration troubleshooting and identification of potential problems or threats to the network. Finally, the ability to export displayed networks as a GraphML file facilitates the virtualization of any network, either planned or actual, via the virtualization portion of the MAVNATT system.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. TESTING AND EVALUATION

A. OVERVIEW OF TESTING

We focused testing efforts on the two primary functions for the mapping application: correct validation of the network plan, and accurate scanning of the live network. To do so, we established a physical test network environment for use while evaluating the live network scan functionality of the application. We then generated different versions of a network plan, each representing this test network. Variations in these network plan versions created specific scenarios that allowed us to test individual sources of potential errors in the mapping application.

B. TEST NETWORK ENVIRONMENT

For consistency and controlled variability during network scanning, we established a test network environment with physical hardware similar to that used by Marine Corps units on LANs. The network consisted of five Cisco 2811 routers, two Cisco 1900 series layer 2 switches, six client computers and a server computer (see Figure 27). The routers were configured using the Enhanced Interior Gateway Routing Protocol. Five of the client computers were loaded with the Ubuntu version 14.04 OS and the sixth (Client3) was loaded with Windows 10. The server computer was configured as both a Domain Name System (DNS) server and a web server on the Ubuntu 14.04 OS. Scripts running on all six client computers were used to conduct periodic DNS queries and to request pages from the web server. These scripts generated a steady flow of traffic and provided a more realistic network scenario.

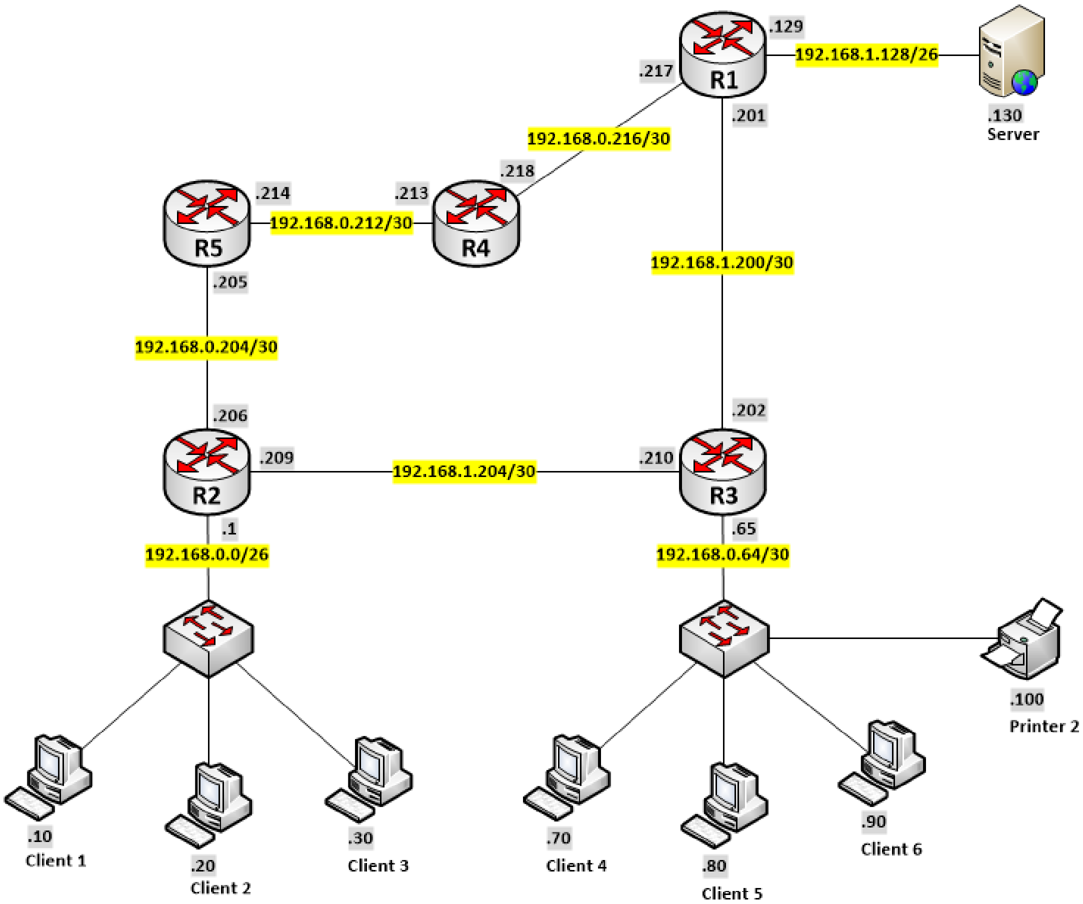


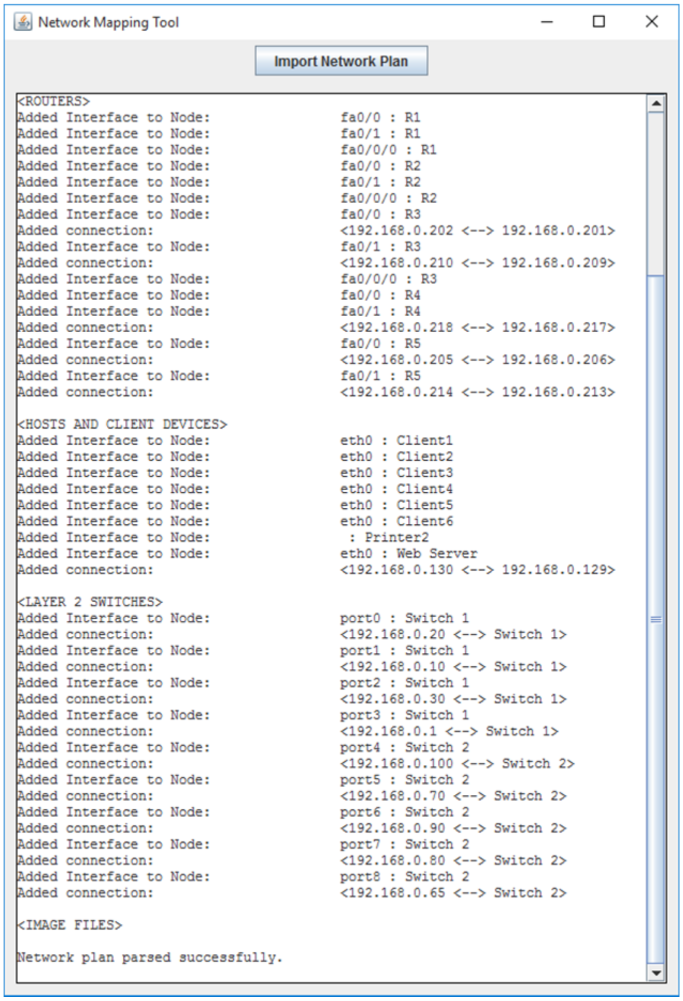
Figure 27. Test Network Environment Configuration.

C. TEST NETWORK PLANS AND CONFIGURATIONS

We generated four variations of the network plan representing the test network environment. Each variation was designed to test key functional points of the mapping application by forcing errors that would then be annotated by the application. For brevity, the image file details for each networked device were omitted from network plans during testing. These details are only relevant for later use within the overall MAVNATT framework. The complete network plan used for each test scenario can be found in Appendix D through Appendix G.

1. Control Configuration

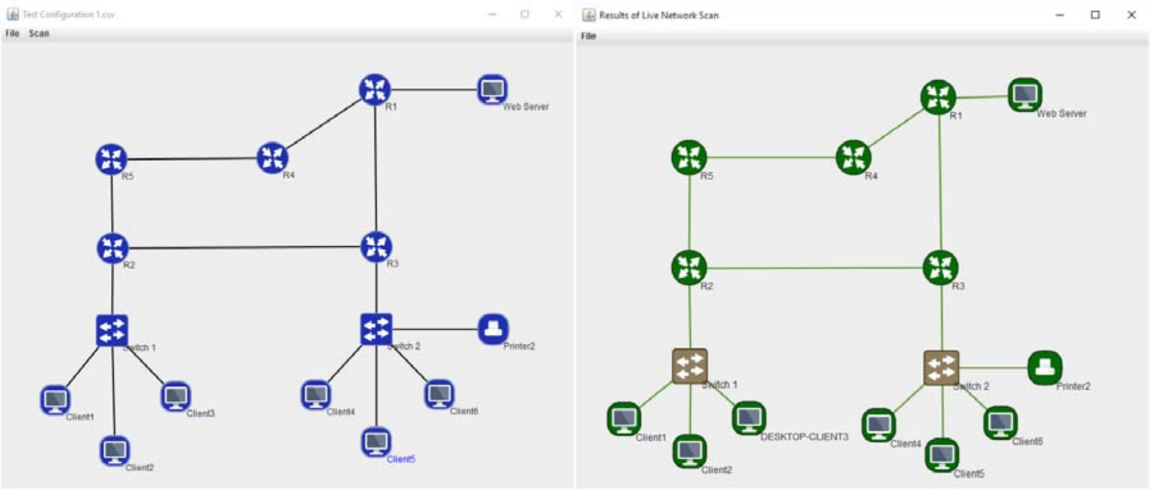
The control configuration network plan (see Appendix D) accurately reflects the actual configuration of the test network shown in Figure 27. Each networked device is correctly represented with its corresponding IP address, MAC address, and next-hop connection. The subnet allocation scheme is valid and all subnets are properly represented on the plan. After importing the network plan, the parse log (see Figure 28) shows no indication of any potential problems.



```
<ROUTERS>
Added Interface to Node:                fa0/0 : R1
Added Interface to Node:                fa0/1 : R1
Added Interface to Node:                fa0/0/0 : R1
Added Interface to Node:                fa0/0 : R2
Added Interface to Node:                fa0/1 : R2
Added Interface to Node:                fa0/0/0 : R2
Added Interface to Node:                fa0/0 : R3
Added connection:                       <192.168.0.202 <--> 192.168.0.201>
Added Interface to Node:                fa0/1 : R3
Added connection:                       <192.168.0.210 <--> 192.168.0.209>
Added Interface to Node:                fa0/0/0 : R3
Added Interface to Node:                fa0/0 : R4
Added Interface to Node:                fa0/1 : R4
Added connection:                       <192.168.0.218 <--> 192.168.0.217>
Added Interface to Node:                fa0/0 : R5
Added connection:                       <192.168.0.205 <--> 192.168.0.206>
Added Interface to Node:                fa0/1 : R5
Added connection:                       <192.168.0.214 <--> 192.168.0.213>

<HOSTS AND CLIENT DEVICES>
Added Interface to Node:                eth0 : Client1
Added Interface to Node:                eth0 : Client2
Added Interface to Node:                eth0 : Client3
Added Interface to Node:                eth0 : Client4
Added Interface to Node:                eth0 : Client5
Added Interface to Node:                eth0 : Client6
Added Interface to Node:                 : Printer2
Added Interface to Node:                eth0 : Web Server
Added connection:                       <192.168.0.130 <--> 192.168.0.129>

<LAYER 2 SWITCHES>
Added Interface to Node:                port0 : Switch 1
Added connection:                       <192.168.0.20 <--> Switch 1>
Added Interface to Node:                port1 : Switch 1
Added connection:                       <192.168.0.10 <--> Switch 1>
Added Interface to Node:                port2 : Switch 1
Added connection:                       <192.168.0.30 <--> Switch 1>
Added Interface to Node:                port3 : Switch 1
Added connection:                       <192.168.0.1 <--> Switch 1>
Added Interface to Node:                port4 : Switch 2
Added connection:                       <192.168.0.100 <--> Switch 2>
Added Interface to Node:                port5 : Switch 2
Added connection:                       <192.168.0.70 <--> Switch 2>
Added Interface to Node:                port6 : Switch 2
Added connection:                       <192.168.0.90 <--> Switch 2>
Added Interface to Node:                port7 : Switch 2
Added connection:                       <192.168.0.80 <--> Switch 2>
Added Interface to Node:                port8 : Switch 2
Added connection:                       <192.168.0.65 <--> Switch 2>

<IMAGE FILES>

Network plan parsed successfully.
```

Figure 28. Control Configuration Network Plan Parse Log.

After conducting the scan of the live network, both the network plan visualization and the live network scan visualization (see Figure 29) match expectations of the test network configuration. All devices on the network are present with details confirmed according to the criteria described in Chapter III, Section C.



Comparison of the network plan visualization (left) and the live network scan visualization (right) shows consistency in results based on the expected value.

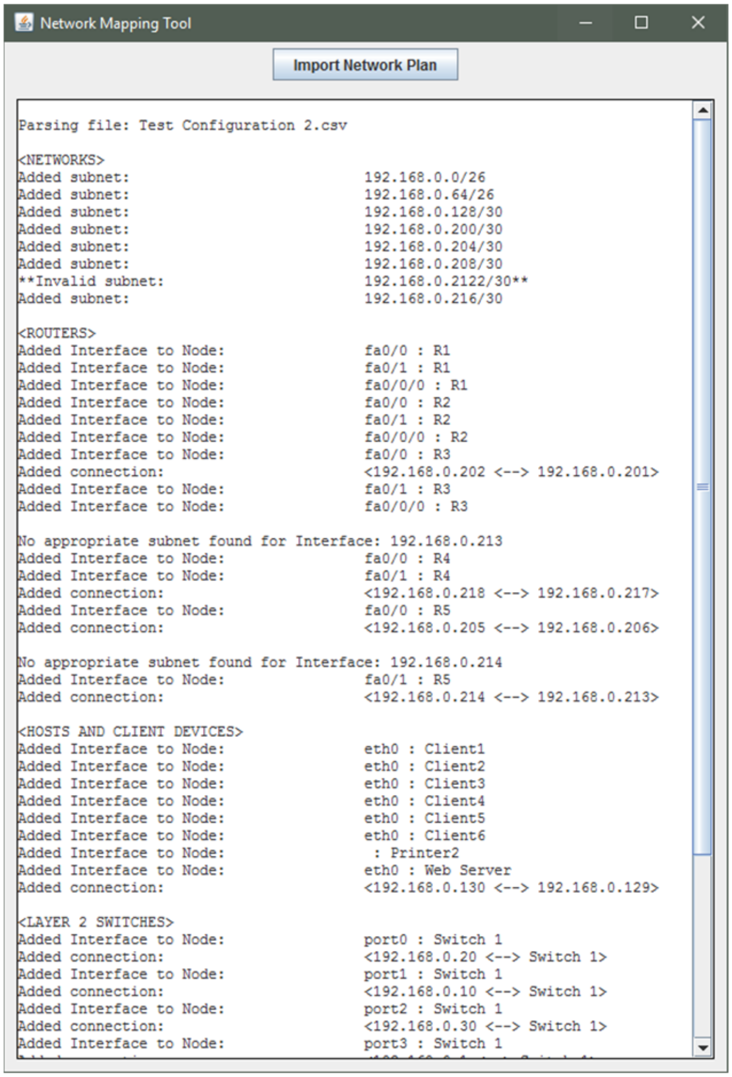
Figure 29. Control Configuration Visualizations.

2. Invalid Network Plan Testing

We developed an invalid network plan (see Appendix E) to test the validation and visualization of the network plan. Inconsistencies and errors were introduced to the plan to test whether the mapping application correctly identified the errors and conveyed the details to the user. These errors represent common mistakes made by network administrators such as typographic errors, incorrect subnet and IP address allocation, or simply misrepresenting the expected configuration in the network plan. The invalid network plan contains the following errors:

- Subnet 192.168.0.212/30 is incorrectly entered as 192.168.0.2122/30, a potential typographic error.
- The next hop address for R3 interface fa0/1 is listed as 192.168.0.211 where it should be 192.168.0.209. The IP address 192.168.0.211 does not exist on the network. This error could be the result of incorrect subnet calculation or simply swapping potential values on the plan.
- The next hop address for R3 interface fa0/0/0 is listed as Switch 3, but no Switch 3 exists on the network.

After importing the network plan, the parse log (see Figure 30) correctly displays a problem with the invalid subnet 192.168.0.2122/30. This subnet is therefore never created within the network framework. Additionally, error messages are reported when adding the interfaces with addresses 192.168.0.213 and 192.168.0.214, because no valid subnet is found that contains these addresses.



```
Parsing file: Test Configuration 2.csv

<NETWORKS>
Added subnet:                           192.168.0.0/26
Added subnet:                           192.168.0.64/26
Added subnet:                           192.168.0.128/30
Added subnet:                           192.168.0.200/30
Added subnet:                           192.168.0.204/30
Added subnet:                           192.168.0.208/30
**Invalid subnet:                       192.168.0.2122/30**
Added subnet:                           192.168.0.216/30

<ROUTERS>
Added Interface to Node:                fa0/0 : R1
Added Interface to Node:                fa0/1 : R1
Added Interface to Node:                fa0/0/0 : R1
Added Interface to Node:                fa0/0 : R2
Added Interface to Node:                fa0/1 : R2
Added Interface to Node:                fa0/0/0 : R2
Added Interface to Node:                fa0/0 : R3
Added connection:                       <192.168.0.202 <--> 192.168.0.201>
Added Interface to Node:                fa0/1 : R3
Added Interface to Node:                fa0/0/0 : R3

No appropriate subnet found for Interface: 192.168.0.213
Added Interface to Node:                fa0/0 : R4
Added Interface to Node:                fa0/1 : R4
Added connection:                       <192.168.0.218 <--> 192.168.0.217>
Added Interface to Node:                fa0/0 : R5
Added connection:                       <192.168.0.205 <--> 192.168.0.206>

No appropriate subnet found for Interface: 192.168.0.214
Added Interface to Node:                fa0/1 : R5
Added connection:                       <192.168.0.214 <--> 192.168.0.213>

<HOSTS AND CLIENT DEVICES>
Added Interface to Node:                eth0 : Client1
Added Interface to Node:                eth0 : Client2
Added Interface to Node:                eth0 : Client3
Added Interface to Node:                eth0 : Client4
Added Interface to Node:                eth0 : Client5
Added Interface to Node:                eth0 : Client6
Added Interface to Node:                 : Printer2
Added Interface to Node:                eth0 : Web Server
Added connection:                       <192.168.0.130 <--> 192.168.0.129>

<LAYER 2 SWITCHES>
Added Interface to Node:                port0 : Switch 1
Added connection:                       <192.168.0.20 <--> Switch 1>
Added Interface to Node:                port1 : Switch 1
Added connection:                       <192.168.0.10 <--> Switch 1>
Added Interface to Node:                port2 : Switch 1
Added connection:                       <192.168.0.30 <--> Switch 1>
Added Interface to Node:                port3 : Switch 1
```

Figure 30. Invalid Network Plan Parse Log.

The network plan visualization (see Figure 31) correctly shows that the expected connections between R2 and R3, and between R3 and Switch 2 are missing. These inconsistencies provide a cue for the network administrator to review his plan for potential errors with these connection settings.

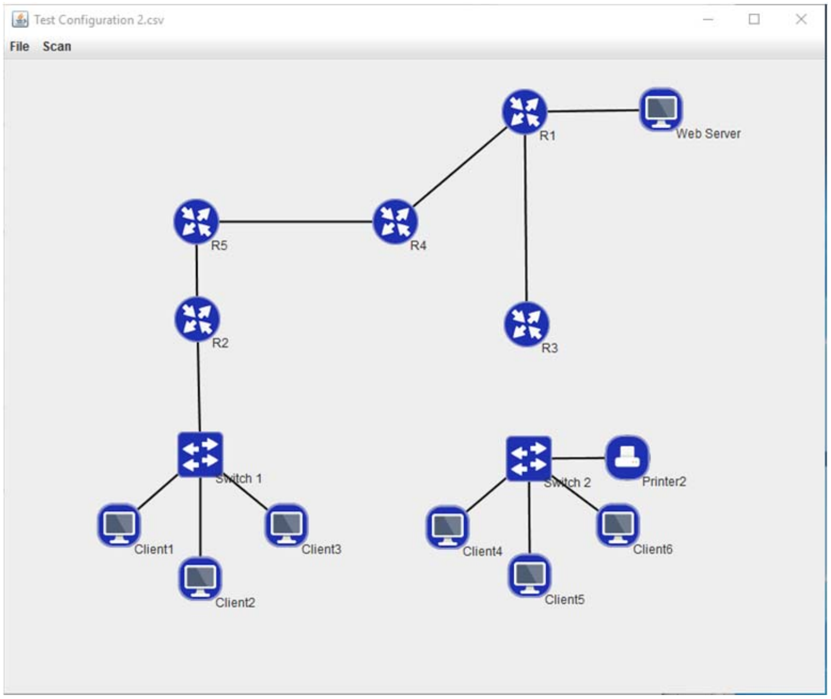


Figure 31. Invalid Network Plan Visualization.

3. Incorrect Network Details Testing

We developed an incorrect network plan (see Appendix F) to test the live network scan functionality of the mapping application. The plan lists incorrect details for some devices on the network to test whether the application correctly identified deviations between the network plan and the live network and conveyed the details of errors to the user. The errors in this test are limited to incorrect MAC addresses and IP addresses for interfaces on the network. The incorrect network plan contains the following errors:

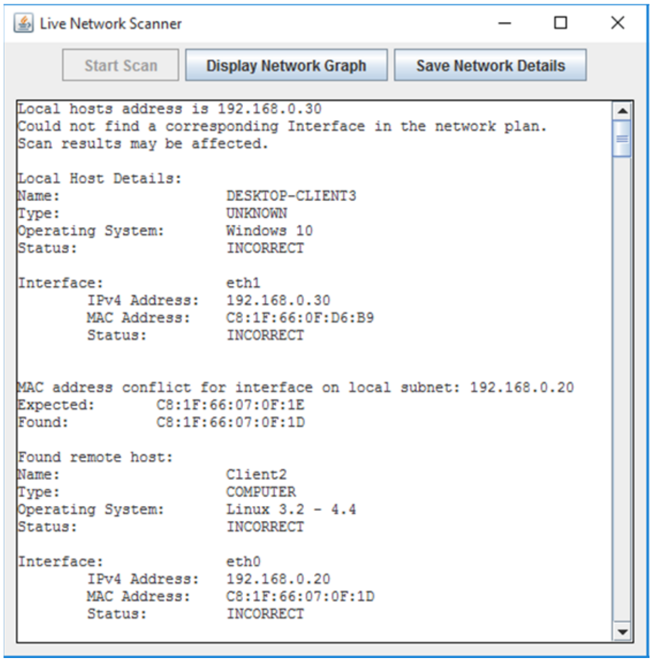
- The next hop address for R3 interface fa0/1 is listed as 192.168.0.211 where it should be 192.168.0.209.
- The IP address for the Client3 interface is listed as 192.168.0.33 instead of the correct value of 192.168.0.30. We ran the mapping application from Client3, so this error affects the details of the local host.

- The MAC address for the Client2 interface is listed as C8:1F:66:07:0F:1E instead of the correct value of C8:1F:66:07:0F:1D.

- The IP address for the Client5 interface is listed as 192.168.0.88 instead of the correct value of 192.168.0.80.

- The MAC address for the Printer2 interface is listed as 00:15:99:D5:5D:33 instead of the correct value of 00:15:99:D5:5D:34.

After importing the incorrect network plan, we scanned the live network, producing the scan console log output shown in Figure 32.



```
Local hosts address is 192.168.0.30
Could not find a corresponding Interface in the network plan.
Scan results may be affected.

Local Host Details:
Name:                       DESKTOP-CLIENT3
Type:                       UNKNOWN
Operating System:           Windows 10
Status:                     INCORRECT

Interface:                  eth1
        IPv4 Address:   192.168.0.30
        MAC Address:    C8:1F:66:0F:D6:B9
        Status:         INCORRECT


MAC address conflict for interface on local subnet: 192.168.0.20
Expected:       C8:1F:66:07:0F:1E
Found:          C8:1F:66:07:0F:1D

Found remote host:
Name:                       Client2
Type:                       COMPUTER
Operating System:           Linux 3.2 - 4.4
Status:                     INCORRECT

Interface:                  eth0
        IPv4 Address:   192.168.0.20
        MAC Address:    C8:1F:66:07:0F:1D
        Status:         INCORRECT
```

Figure 32. Incorrect Network Details Scan Log.

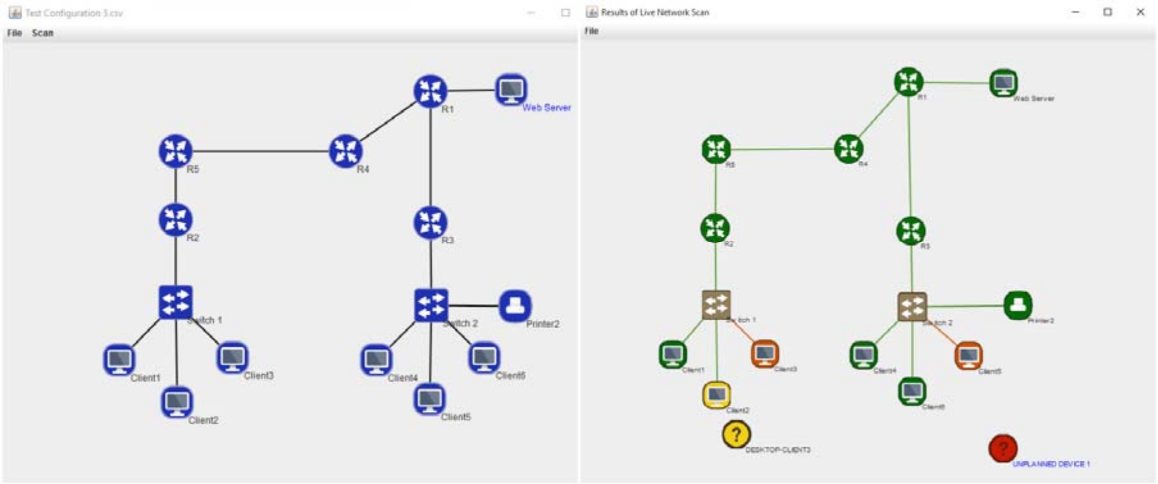
The scan console log correctly annotates the following conflicts:

- The application identified a local host addressing conflict. The local host IP was identified as 192.168.0.30, but there is no interface with a corresponding address represented on the network plan. A new Node is created to represent the actual local host.

- The application discovered a host at Client2's expected IP address, but with a MAC address that is not the expected value represented on the network plan. This IP address is on the same subnet as the local host, making MAC address verification possible. The status of Client2 is set as "INCORRECT."

- The application failed to locate any device at IP address 192.168.0.33, so the status of Client3 is set to “MISSING.”
- The application failed to locate any device at IP address 192.168.0.88, so the status of Client5 is set to “MISSING.”
- The application discovered an unexpected device at IP address 192.168.0.80. This is the actual address of Client5. A new Node object is created and added to the network to represent the unexpected device.

The visualizations of the planned network and the live network scan results (see Figure 33) reflect the errors annotated in the scan console log. A new Node object with the correct values of the local host is added to the network, represented in yellow. Client2 is yellow to denote the incorrect MAC address. Client3 and Client5 are orange to represent that they are on the network plan, but were not found during the live network scan. A newly discovered device (the actual Client5) is shown in red. Additionally, the expected connection between R2 and R3 is again missing, which represents the incorrect value given for the next hop destination of R3 interface fa0/0/0. This connection actually exists on the network but the mapping application has no way to detect or verify it.



Comparison of the network plan visualization (left) and the live network scan visualization (right) shows several discrepancies.

Figure 33. Incorrect Network Details Visualizations.

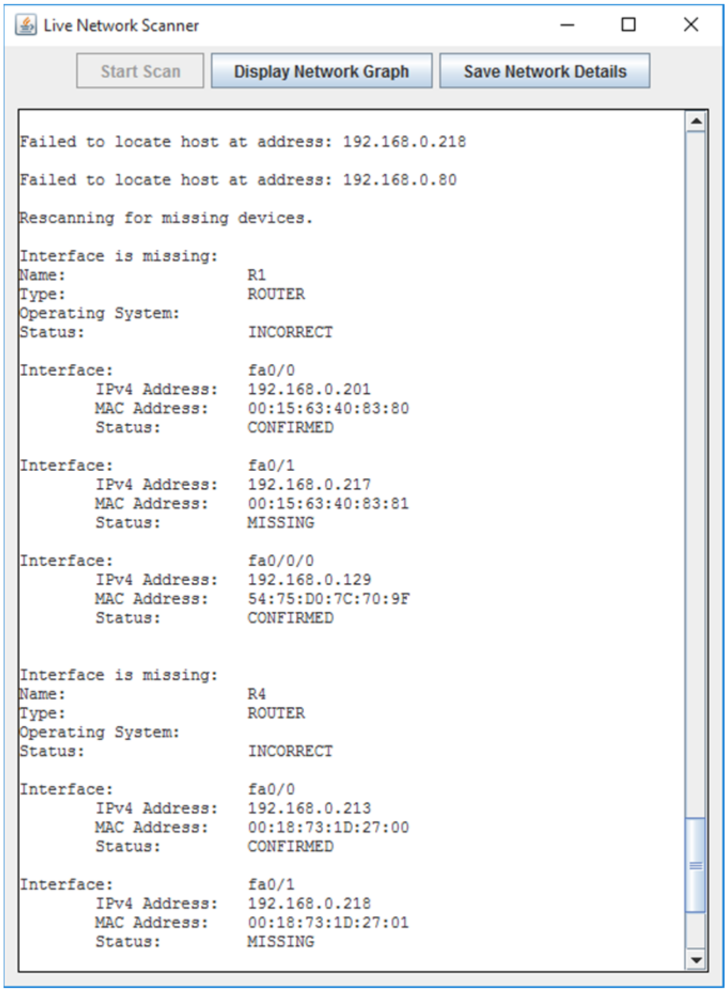
A notable point in this result is that Printer2 is green and listed as confirmed, even though the MAC address was listed incorrectly in the network plan. This is the expected behavior of the application. The live network scan was conducted from Client3, on a different subnet than Printer2. The mapping application has no access to the MAC addresses of devices on remote subnets and therefore confirms remote devices by matching IP addresses only.

4. Invalid Network Configuration Testing

For the final test, we focused on inconsistencies in the test network rather than incorrect data entry in the network plan. This test was developed to assess the live network scan functionality of the mapping application. In the test, some devices were turned off and cables were disconnected. These errors represent device outages or other common problems when managing a live network. The network plan itself (see Appendix G) is largely the same as the control configuration network plan with a single exception. The variations created for this test are as follows:

- Subnet 192.168.0.208/30 and 192.168.0.12/30 are merged into a single subnet: 192.168.0.208/29. Given the intended network configuration, this creates an invalid IP addressing scheme because there are multiple collision domains using the same subnet.
- The connection between R1 interface fa0/1 and R4 interface fa0/1 was unplugged.
- Client5 was turned off.

After importing the invalid configuration network plan, the parse log shows no indication of any problems, as expected. We then scanned the live network, producing the scan console log output shown in Figure 34. The mapping application correctly identifies that R1 interface fa0/1 and R4 interface fa0/1 are unreachable as a result of the unplugged cable. The Edge object representing the connection between these two interfaces has been set to “MISSING.” The application also identified that Client5 was unreachable, the result of that computer being turned off.



```
Failed to locate host at address: 192.168.0.218

Failed to locate host at address: 192.168.0.80

Rescanning for missing devices.

Interface is missing:
Name:                   R1
Type:                   ROUTER
Operating System:
Status:                 INCORRECT

Interface:              fa0/0
        IPv4 Address:   192.168.0.201
        MAC Address:    00:15:63:40:83:80
        Status:         CONFIRMED

Interface:              fa0/1
        IPv4 Address:   192.168.0.217
        MAC Address:    00:15:63:40:83:81
        Status:         MISSING

Interface:              fa0/0/0
        IPv4 Address:   192.168.0.129
        MAC Address:    54:75:D0:7C:70:9F
        Status:         CONFIRMED


Interface is missing:
Name:                   R4
Type:                   ROUTER
Operating System:
Status:                 INCORRECT

Interface:              fa0/0
        IPv4 Address:   192.168.0.213
        MAC Address:    00:18:73:1D:27:00
        Status:         CONFIRMED

Interface:              fa0/1
        IPv4 Address:   192.168.0.218
        MAC Address:    00:18:73:1D:27:01
        Status:         MISSING
```

Figure 34. Invalid Network Configuration Scan Log.

The visualization of the live network scan results (see Figure 35) reflects the errors annotated in the scan console log. In addition to identifying the missing device, the mapping application correctly identifies and displays errors in the connections between devices. The edge representing the connection between R1 interface fa0/1 and R4 interface fa0/1 has been set to "MISSING" and is thus colored orange. The edges between R2 interface fa0/1 and R3 interface fa0/1 and between R4 interface fa0/0 and R5 interface fa0/1 are both set as "INCORRECT" and colored yellow. This indicates a potential problem with the IP addressing scheme where the interfaces on these edges are expected to be on the same subnet. This is an invalid network configuration.

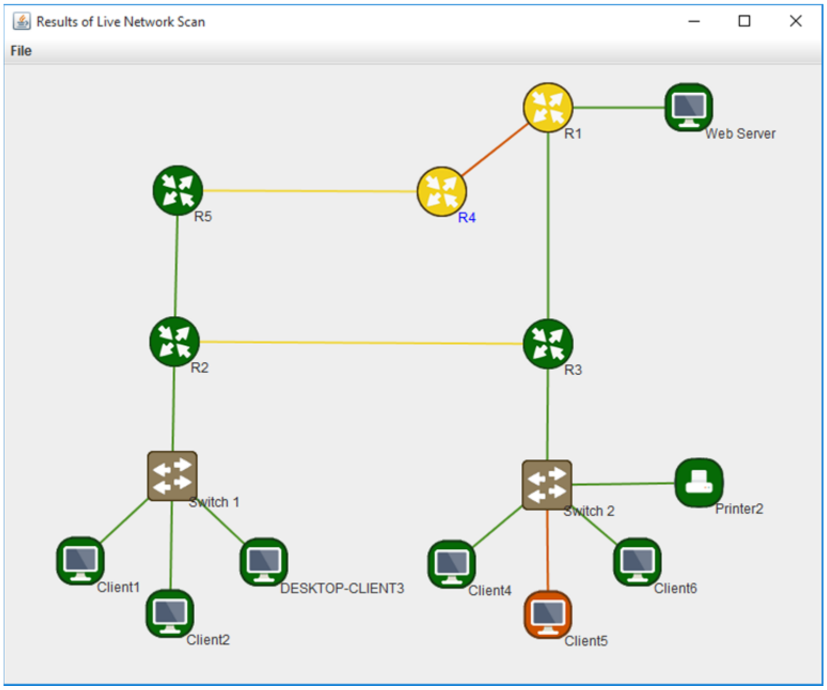


Figure 35. Invalid Network Configuration Scan Visualization.

D. SUMMARY

This chapter details the testing and evaluation conducted on the MAVNATT mapping module application. We established a physical test network that closely resembles those in use by Marine Corps units to provide a consistent and modifiable environment on which to run test scenarios. Four separate network plans and configurations were developed to test specific functional points of the mapping application. These test scenarios showed that the mapping application functions as expected and correctly identifies potential problems in a network plan as well as conflicts between the network plan and the actual live network configuration. Results from these tests show that the live network scan results combined with the network plan visualization provide a useful tool for network administrators to troubleshoot networks.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION AND FUTURE WORK

A. CONCLUSION

In this thesis, we have successfully developed a network mapping application for integration into the proposed MAVNATT system. This application provides functionality for network administrators to develop a network plan, visualize and verify that plan, scan the live network for comparison and validation, and export the network configuration for import and use by the MAVNATT awareness and virtualization modules. These functions of the mapping application provide the necessary foundation and starting point to virtualize an accurate replica of a local area network for use in training and security applications.

The proposed network plan format allows the network administrator to succinctly and accurately define all entities on a network. The XML-style tagging provides for extensibility in the future, as more tags can be defined and implemented to increase the level of detail or introduce new elements to the network configuration. The decision to generate the plan in Microsoft Excel facilitates ease of use and portability across a broad range of computers due to the popularity and widespread use of the program. The plan is easily generated, modified and extended without reliance on complicated database applications or proprietary programs for data storage and manipulation. Most importantly, the .xlsx file format can be exported to a simple text file, allowing the mapping application to import and parse the network plan.

The mapping application uses an object-oriented framework to represent the network architecture in a complete, detailed, and extensible manner. The relational interactions between objects in the network framework correctly mimic hierarchy and interconnectivity of devices on a LAN. This allows for future development, implementation, and extension to provide further functionality, detail and feature sets by extending the individual objects within the framework. Furthermore, this framework provides a foundation for use by other modules with the MAVNATT system to interact with and develop network-related structures.

The visualization feature of the mapping application is an important function that assists the network administrator in verifying the network, allowing for quick recognition of problems and pitfalls. The visual representation of the network plan after its import provides an opportunity to recognize potential errors in the planned configuration and settings. Likewise, the visualization of the live network after scanning provides an easy to use tool, color coded for quick reference and confirmation of actual settings compared to the expected values of the network plan. The visualization screens are reconfigurable, allowing the user to drag, rotate, and shift individual nodes in the graph to a location of their choice. This further facilitates ease of use, allowing network administrators to customize the visualization to their liking.

In addition to the simple visualization view, the application displays log messages to the console throughout the mapping process. These log messages provide further detail on the status of the mapping effort, identifying potential errors and misconfigured settings in the process. When mapping is complete, the user has the option to save a detailed report of the current network configuration and settings as a text file. These log messages and configuration files provide the fine-grained detail needed for the network administrator to correct problems identified during the mapping and visualization process.

Finally, the mapping application provides functionality to export the network in the GraphML file format. This file provides information about the configuration and settings of the network, including details of individual network entities as well as connections between devices. The content of the file can be easily extended, either manually or by making alterations to the mapping application itself. This again provides the necessary framework for the future addition of features including increased detail of network entities. This export operation is a key function, as it provides the necessary information required for other modules within the MAVNATT framework to import a complete network representation.

We believe this mapping application provides a full-featured foundation for the proposed MAVNATT system. The application includes all the functionality needed to correctly and accurately plan, visualize, scan and export a representation of a local area network. This representation can be used for analysis or for further import by another module in the MAVNATT system.

B. FUTURE WORK

This application provides the basic functionality needed to satisfy the requirements of the MAVNATT mapping module; however, there are areas for further research as well as the implementation of increased functionality. Enhancements to the mapping application itself can be made to increase the detail available for individual network entities, provide an improved feature set for the application, or to allow for increased flexibility within the network plan. Further work is also required to fully implement, test, and integrate the mapping module into the MAVNATT framework to provide a start-to-finish functionality of mapping, monitoring, and virtualizing a network.

1. Increased Detail and Flexibility

The mapping application in its current state provides a limited amount of detail for representing a LAN. Individual devices are identified by the MAC address and IP address, with the ability to store additional data such as device name, gateway router, etc. Future implementations can expand on this foundation to provide an increased level of detail and fidelity, allowing for greater depth when representing a network. Advanced methods such as device fingerprinting and connection-oriented data collection can be used to increase the level of fidelity for identification of networked devices.

a. Device Fingerprinting

A device fingerprinting approach would provide functionality to uniquely identify specific machines on the network with a higher level of fidelity. Such an approach would require a network administrator to generate an initial database of fingerprints, either for individual devices on the network, or for specific subsets of devices with particular services. After the initial establishment of the fingerprint database, the live network scan can confirm or deny the presence of a networked device via the stored identifying information rather than just the MAC address or IP address.

b. Connection-Oriented Data Collection

The mapping application currently uses probing techniques to retrieve data about networked devices. The application sends a probe to specific addresses, eliciting an automatic response from the network stack of that device without actually making a connection. This response is then parsed for required information. A connection-oriented approach would allow the user to collect more in-depth information about specific devices. Options for initiating connections to remote devices include Secure Shell for routers, or the establishment of Simple Network Management Protocol relationships between the devices on the network.

c. Dynamic Network Configurations

The devised network plan format described in Chapter III, Section C, requires static assignment of IP addresses for all devices on the network. This format presents complications when mapping LANs that implement the Dynamic Host Configuration Protocol to automatically assign IP addresses. Such implementation is common with wireless networks or those that have devices connected intermittently. Future improvements to the mapping application could account for the dynamic nature of such networks. Doing so requires identifying devices on the network by features other than IP address alone, such as the fingerprinting method described above. Another option would be run a mapping application from a local host on each subnet in the LAN; however, further research and development is needed to correlate that data and scan results from multiple applications.

2. Mapping Unknown Networks

This implementation of the MAVNATT mapping module assumes administrator-level access to a LAN along with detailed prior knowledge of the configuration and topology of the network. There is currently no functionality for mapping an unknown network from an unknown starting point. Further research and extension of the application could potentially address this issue. Mapping a network from an unknown starting point presents certain difficulties, and in many cases a topology cannot be confirmed, only estimated. Extension of the mapping application to include the functionality to scan unknown networks could potentially lead to a tool to determine information about adversarial networks.

a. Bounded Search Requirement

One difficulty when mapping an unknown network is the question of where to stop. If a LAN consists of several subnets, then any effort to map the entire LAN must necessarily send traffic across broadcast domains to reach devices separated by routers. Without prior knowledge of the subnet and IP address allocation schemes, automated mapping efforts could easily extend search areas to those beyond the LAN in question. Methods to remedy this problem include enforcing a bounded search constraint by limiting the hop count of probes from the source device, and artificially making inferences about the subnet to be search. In either case, it is impossible to concretely verify that the complete composition of a LAN has been discovered without at least some prior knowledge of that LAN. Future research could investigate the limitations and accuracy of an unbounded search in order to determine whether an automated application is a viable option for mapping completely unknown networks.

b. Connection Information Discovery

The automatic discovery of device-to-device connection information within a network represents a significant challenge. The implementation of the mapping application in this thesis assumes that a planned connection is correct if both endpoints of that connection have been confirmed; however, the actual physical connection topology may deviate from the expected planned configuration. A subnet and its broadcast domain may actually be subdivided by intermediate devices (e.g., hubs and switches). These devices forward traffic on the data link layer and are virtually transparent on the network layer; however, they also contain valuable information about the network topology via the internal switch table which contains a MAC address listing of all connected devices. Discovery of these devices and their corresponding connections would require additional techniques for monitoring physical layer and data link layer traffic in conjunction with network layer traffic, but will yield a much more robust picture of the network topology.

c. Point of Collection

The amount and type of traffic available for analysis varies greatly with the point of collection on the network, of which there are several options. Data collection can be conducted from a host computer connected to the subnet, which is the method employed by the mapping application in this thesis. Alternatively, a wiretap device can be used to passively collect traffic from a point within the network. Both options allow for collection and analysis of local network traffic, but both require physical access to the network. In certain scenarios, such as mapping an adversarial network, physical access is not a viable option. Mapping from within a network is fundamentally different from mapping from an external position. Efforts to map an external network can be constrained by several factors including network address translation, firewalls, and security policies which may hinder mapping efforts using typical scanning techniques.

3. Integration, Testing, and Utilization

As of publication of this thesis, the MAVNATT system has not been fully implemented or integrated. In addition to the mapping application proposed here, a solution for the virtualization module has been proposed by Naval Postgraduate School student, Erik Berndt [22]. Further research on the implementation of the proposed awareness module, as well as integration of the three modules into the overall MAVNATT framework, is required.

a. MAVNATT Awareness Module Development

The goal of the proposed awareness module is to monitor a live network and provide the network administrator with real-time updates of changes to the status of the network. This can be accomplished by using the network framework described in Chapter III, Section B, to accurately represent the live network in its current state. An application to passively monitor network traffic can compare information found in packet headers and payloads to that of the expected values of the mapped network. This functionality is similar to that of the mapping application; an awareness module might be implemented by extending the functionality of the mapping application to provide a periodic or continuous feedback loop.

b. Module Integration into the MAVNATT Framework

Further research is required to integrate the mapping module into the proposed MAVNATT framework and to conduct a complete process of replicating a live network starting from a network plan. The network plan used as input to the mapping application and the file formats available for export have been carefully developed to allow for portability and extensibility. In particular, the GraphML file format contains all required information to represent the network to a sufficient level of detail to create a virtualized replica of that network. In theory, the awareness and virtualization modules can import the GraphML file generated by the mapping application; however, further refinements may be required to ensure compatibility and test functionality.

c. MAVNATT Testing

Upon completion of development and integration of all MAVNATT modules, research is required to test functionality and limitations of the overall system. The desired end state is that a user can generate and import a network plan, scan the live network to compare findings, monitor the network based on those finding, and create a virtualized replica of the network. Testing is required to determine limitations on the type and number of devices that can be virtualized and any potential errors when virtualizing planned networks.

d. Utilization of the MAVNATT System

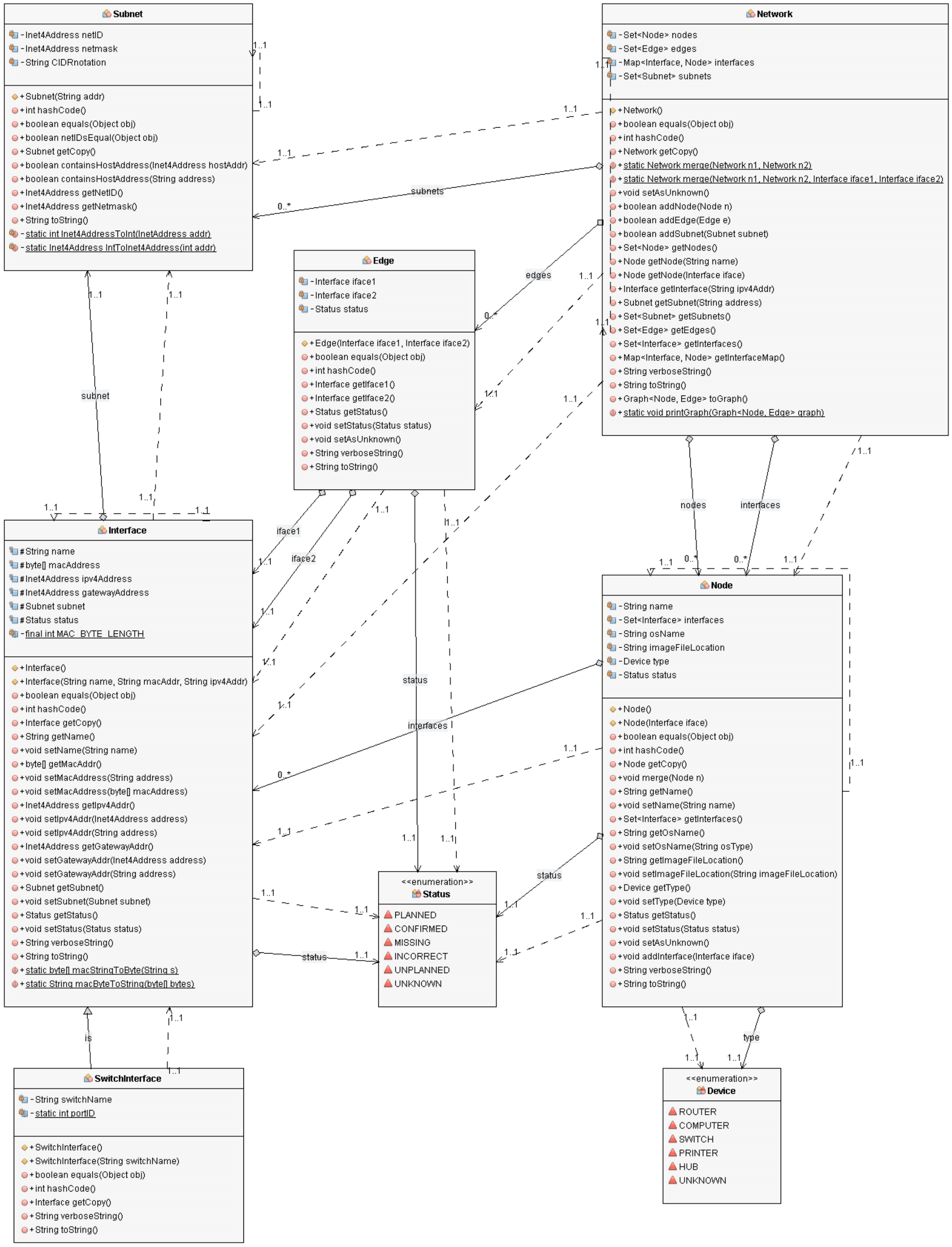
The initially proposed purpose of the MAVNATT system was to allow network administrators a training tool for evaluating their local network without fear of disrupting the live network stability. The functionality of MAVNATT to plan and virtualize practically any network configuration provides the opportunity to use it in scenarios other than for network administrator training, such as those employed on cyber ranges. A cyber range is a fully functional network environment used for testing and training purposes that extend beyond the scope of general network administration. Typical cyber range usage scenarios include:

- Penetration testing to determine vulnerabilities in the network configuration and how to defend against exploitation.

- Simulation of effects of malware or viruses on the network.
- Training for the offensive exploitation of network vulnerabilities.

Cyber ranges are generally set up as physical networks, limiting their availability and scale of the network topology. The ability to dynamically configure and virtualize a network with the MAVNATT system could potentially provide a new method for employing cyber ranges. Extended cyber ranges could also be created by connecting two or more virtualized networks, or by connecting a virtualized network to a physical cyber range.

APPENDIX A. NETWORK FRAMEWORK UML DIAGRAM


Subnet
-Inet4Address netID
-Inet4Address netmask
-String CIDRnotation
+Subnet(String addr)
+int hashCode()
+boolean equals(Object obj)
+boolean netIDsEqual(Object obj)
+Subnet getCopy()
+boolean containsHostAddress(Inet4Address hostAddr)
+boolean containsHostAddress(String address)
+Inet4Address getNetID()
+Inet4Address getNetmask()
+String toString()
-static int Inet4AddressToInt(InetAddress addr)
-static Inet4Address IntToInet4Address(int addr)
Network
-Set<Node> nodes
-Set<Edge> edges
-Map<Interface, Node> interfaces
-Set<Subnet> subnets
+Network()
+boolean equals(Object obj)
+int hashCode()
+Network getCopy()
+static Network merge(Network n1, Network n2)
+static Network merge(Network n1, Network n2, Interface iface1, Interface iface2)
+void setAsUnknown()
+boolean addNode(Node n)
+boolean addEdge(Edge e)
+boolean addSubnet(Subnet subnet)
+Set<Node> getNodes()
+Node getNode(String name)
+Node getNode(Interface iface)
+Interface getInterface(String ipv4Addr)
+Subnet getSubnet(String address)
+Set<Subnet> getSubnets()
+Set<Edge> getEdges()
+Set<Interface> getInterfaces()
+Map<Interface, Node> getInterfaceMap()
+String verboseString()
+String toString()
+Graph<Node, Edge> toGraph()
+static void printGraph(Graph<Node, Edge> graph)
Edge
-Interface iface1
-Interface iface2
-Status status
+Edge(Interface iface1, Interface iface2)
+boolean equals(Object obj)
+int hashCode()
+Interface getIface1()
+Interface getIface2()
+Status getStatus()
+void setStatus(Status status)
+void setAsUnknown()
+String verboseString()
+String toString()
Interface
#String name
#byte[] macAddress
#Inet4Address ipv4Address
#Inet4Address gatewayAddress
#Subnet subnet
#Status status
-final int MAC_BYTE_LENGTH
+Interface()
+Interface(String name, String macAddr, String ipv4Addr)
+boolean equals(Object obj)
+int hashCode()
+Interface getCopy()
+String getName()
+void setName(String name)
+byte[] getMacAddr()
+void setMacAddress(String address)
+void setMacAddress(byte[] macAddress)
+Inet4Address getIpv4Addr()
+void setIpv4Addr(Inet4Address address)
+void setIpv4Addr(String address)
+Inet4Address getGatewayAddr()
+void setGatewayAddr(Inet4Address address)
+void setGatewayAddr(String address)
+Subnet getSubnet()
+void setSubnet(Subnet subnet)
+Status getStatus()
+void setStatus(Status status)
+String verboseString()
+String toString()
+static byte[] macStringToByte(String s)
+static String macByteToString(byte[] bytes)
Node
-String name
-Set<Interface> interfaces
-String osName
-String imageFileLocation
-Device type
-Status status
+Node()
+Node(Interface iface)
+boolean equals(Object obj)
+int hashCode()
+Node getCopy()
+void merge(Node n)
+String getName()
+void setName(String name)
+Set<Interface> getInterfaces()
+String getOsName()
+void setOsName(String osType)
+String getImageFileLocation()
+void setImageFileLocation(String imageFileLocation)
+Device getType()
+void setType(Device type)
+Status getStatus()
+void setStatus(Status status)
+void setAsUnknown()
+void addInterface(Interface iface)
+String verboseString()
+String toString()
<<enumeration>>
Status
PLANNED
CONFIRMED
MISSING
INCORRECT
UNPLANNED
UNKNOWN
SwitchInterface
-String switchName
-static int portID
+SwitchInterface()
+SwitchInterface(String switchName)
+boolean equals(Object obj)
+int hashCode()
+Interface getCopy()
+String verboseString()
+String toString()
<<enumeration>>
Device
ROUTER
COMPUTER
SWITCH
PRINTER
HUB
UNKNOWN
subnets
0..*
edges
0..*
subnet
iface1
iface2
status
interfaces
nodes
0..*
status
type
is
1..1

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. NETWORK PLAN CSV FILE

```
<NETWORKS>,,,,,,
Subnet ID,Notes,,,,,
192.168.0.0/26,,,,,,
192.168.0.64/26,,,,,,
192.168.0.208/30,,,,,,
</NETWORKS>,,,,,,
<ROUTERS>,,,,,,
Router Name,Interface Name,Interface MAC,Interface IP,Next Hop (IP or switch name),,
R2,fa0/1,00:1B:D4:EF:5C:89,192.168.0.209,192.168.0.210,,
R2,fa0/0/0,00:1B:D4:EF:5C:88,192.168.0.1,Switch 1,,
R3,fa0/1,00:1B:54:A9:6D:C9,192.168.0.210,192.168.0.209,,
R3,fa0/0/0,58:8D:09:76:E9:F4,192.168.0.65,Switch 2,,
</ROUTERS>,,,,,,
<HOSTS AND CLIENT DEVICES>,,,,,,
Host Name,Interface Name,Interface MAC,Interface IP,Next Hop (IP or switch name),Gateway Address,Device Type
Client2,eth0,C8:1F:66:07:0F:1D,192.168.0.20,Switch 1,192.168.0.1,COMPUTER
Client3,eth0,C8:1F:66:0F:D6:B9,192.168.0.30,Switch 1,192.168.0.1,COMPUTER
Client5,eth0,D8:CB:8A:60:4B:E0,192.168.0.80,Switch 2,192.168.0.65,COMPUTER
Client6,eth0,C8:1F:66:07:10:7E,192.168.0.90,Switch 2,192.168.0.65,COMPUTER
Printer2,ethernet,00:15:99:D5:5D:34,192.168.0.100,Switch 2,192.168.0.65,PRINTER
</HOSTS AND CLIENT DEVICES>,,,,,,
<LAYER 2 SWITCHES>,,,,,,
Switch Name,Number of connections,,,,,
Switch 1,3,,,,,
Switch 2,4,,,,,
</LAYER 2 SWITCHES>,,,,,,
<IMAGE FILES>,,,,,,
Platform Name,Operating System Name,Image File Location,,,,
R2,IOS,C:\Users\M4600\Desktop\MAVNATT\REPO\router2.vdi,,,,
R3,IOS,C:\Users\M4600\Desktop\MAVNATT\REPO\router3.vdi,,,,
Client2,Ubuntu,C:\Users\M4600\Desktop\MAVNATT\REPO\Ubuntu_1.vdi,,,,
Client3,Windows 10,C:\Users\M4600\Desktop\MAVNATT\REPO\Windows10_1.vdi,,,,
Client5,Ubuntu,C:\Users\M4600\Desktop\MAVNATT\REPO\Ubuntu_1.vdi,,,,
Client6,Ubuntu,C:\Users\M4600\Desktop\MAVNATT\REPO\Ubuntu_1.vdi,,,,
</IMAGE FILES>,,,,,,
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. LIVE NETWORK DETAILS FILE

```
Results of live network scan on Thu Apr 07 09:05:06 PDT 2016

==============NETWORK DEVICES==============

Name:                R2
   Type:             ROUTER
   Operating System: IOS
   Status:           CONFIRMED

   Interface:        fa0/1
      IPv4 Address:  192.168.0.209
      MAC Address:   00:1B:D4:EF:5C:89
      Status:        CONFIRMED

   Interface:        fa0/0/0
      IPv4 Address:  192.168.0.1
      MAC Address:   00:1B:D4:EF:5C:88
      Status:        CONFIRMED

Name:                R3
   Type:             ROUTER
   Operating System: IOS
   Status:           CONFIRMED

   Interface:        fa0/0/0
      IPv4 Address:  192.168.0.65
      MAC Address:   58:8D:09:76:E9:F4
      Status:        CONFIRMED

   Interface:        fa0/1
      IPv4 Address:  192.168.0.210
      MAC Address:   00:1B:54:A9:6D:C9
      Status:        CONFIRMED

Name:                Client2
   Type:             COMPUTER
   Operating System: Linux 3.2 - 4.4
   Status:           CONFIRMED

   Interface:        eth0
      IPv4 Address:  192.168.0.20
      MAC Address:   C8:1F:66:07:0F:1D
      Status:        CONFIRMED
```

```
Name:                DESKTOP-CLIENT3
   Type:             COMPUTER
   Operating System: Windows 10
   Status:           CONFIRMED

   Interface:        eth1
      IPv4 Address:  192.168.0.30
      MAC Address:   C8:1F:66:0F:D6:B9
      Status:        CONFIRMED

Name:                Client5
   Type:             COMPUTER
   Operating System: Linux 3.2 - 4.4
   Status:           CONFIRMED

   Interface:        eth0
      IPv4 Address:  192.168.0.80
      MAC Address:   D8:CB:8A:60:4B:E0
      Status:        CONFIRMED

Name:                Client6
   Type:             COMPUTER
   Operating System: Linux 3.2 - 4.4
   Status:           CONFIRMED

   Interface:        eth0
      IPv4 Address:  192.168.0.90
      MAC Address:   C8:1F:66:07:10:7E
      Status:        CONFIRMED

Name:                Printer2
   Type:             PRINTER
   Operating System:
   Status:           CONFIRMED

   Interface:        ethernet
      IPv4 Address:  192.168.0.100
      MAC Address:   00:15:99:D5:5D:34
      Status:        CONFIRMED

Name:                Switch 1
   Type:             SWITCH
   Connections:      3

Name:                Switch 2
   Type:             SWITCH
   Connections:      4
```

```
============NETWORK CONNECTIONS============

<192.168.0.210 <--> 192.168.0.209> : CONFIRMED
<192.168.0.20 <--> Switch 1> : CONFIRMED
<192.168.0.30 <--> Switch 1> : CONFIRMED
<192.168.0.1 <--> Switch 1> : CONFIRMED
<192.168.0.100 <--> Switch 2> : CONFIRMED
<192.168.0.90 <--> Switch 2> : CONFIRMED
<192.168.0.80 <--> Switch 2> : CONFIRMED
<192.168.0.65 <--> Switch 2> : CONFIRMED
```

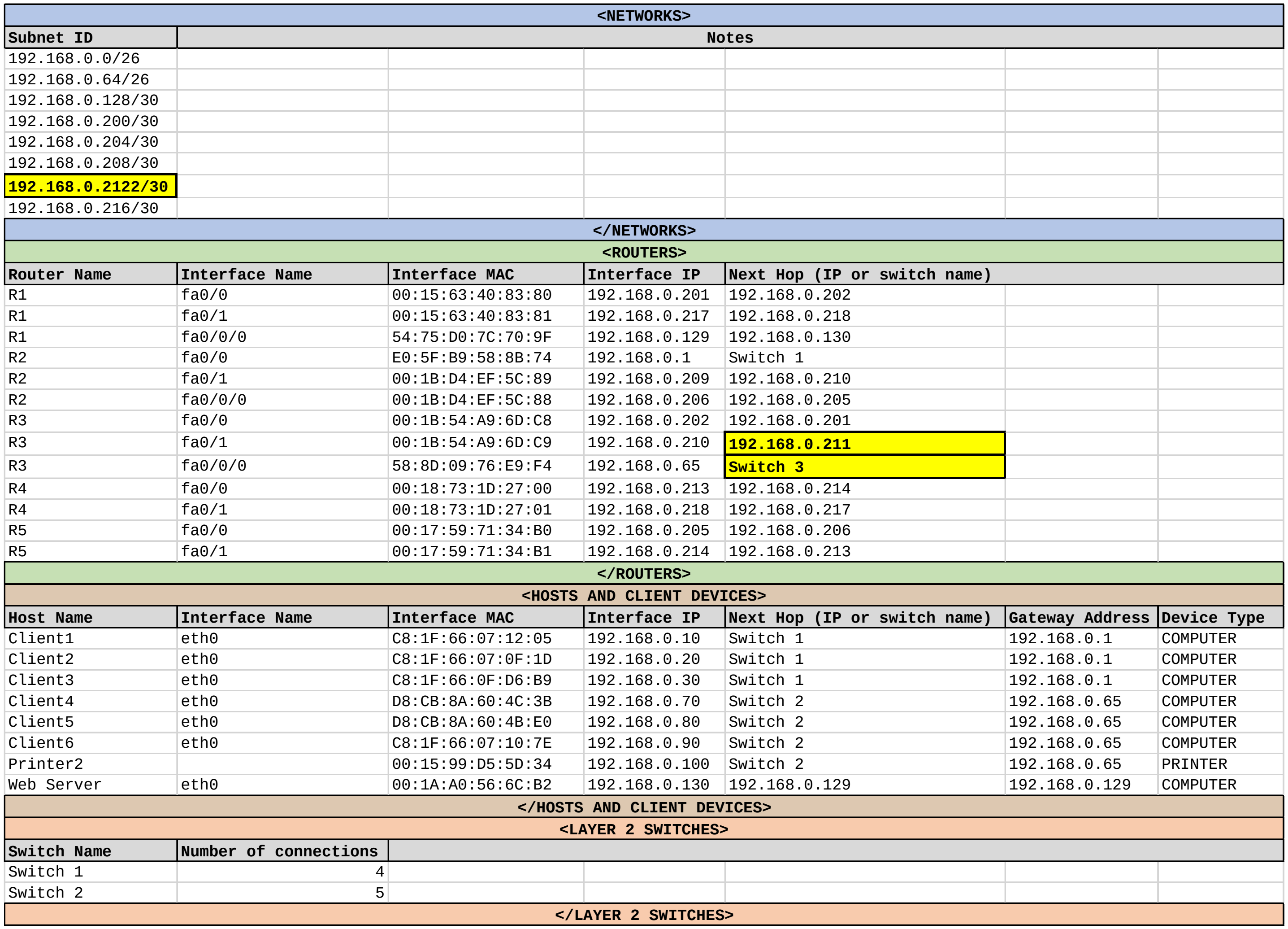
THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. CONTROL CONFIGURATION NETWORK PLAN

<NETWORKS>						
Subnet ID	**Notes**					
192.168.0.0/26						
192.168.0.64/26						
192.168.0.128/30						
192.168.0.200/30						
192.168.0.204/30						
192.168.0.208/30						
192.168.0.212/30						
192.168.0.216/30						
</NETWORKS>						
<ROUTERS>						
Router Name	**Interface Name**	**Interface MAC**	**Interface IP**	**Next Hop (IP or switch name)**		
R1	fa0/0	00:15:63:40:83:80	192.168.0.201	192.168.0.202		
R1	fa0/1	00:15:63:40:83:81	192.168.0.217	192.168.0.218		
R1	fa0/0/0	54:75:D0:7C:70:9F	192.168.0.129	192.168.0.130		
R2	fa0/0	00:1B:D4:EF:5C:88	192.168.0.1	Switch 1		
R2	fa0/1	00:1B:D4:EF:5C:89	192.168.0.209	192.168.0.210		
R2	fa0/0/0	EF:5F:B9:58:8B:74	192.168.0.206	192.168.0.205		
R3	fa0/0	00:1B:54:A9:6D:C8	192.168.0.202	192.168.0.201		
R3	fa0/1	00:1B:54:A9:6D:C9	192.168.0.210	192.168.0.209		
R3	fa0/0/0	58:8D:09:76:E9:F4	192.168.0.65	Switch 2		
R4	fa0/0	00:18:73:1D:27:00	192.168.0.213	192.168.0.214		
R4	fa0/1	00:18:73:1D:27:01	192.168.0.218	192.168.0.217		
R5	fa0/0	00:17:59:71:34:B0	192.168.0.205	192.168.0.206		
R5	fa0/1	00:17:59:71:34:B1	192.168.0.214	192.168.0.213		
</ROUTERS>						
<HOSTS AND CLIENT DEVICES>						
Host Name	**Interface Name**	**Interface MAC**	**Interface IP**	**Next Hop (IP or switch name)**	**Gateway Address**	**Device Type**
Client1	eth0	C8:1F:66:07:12:05	192.168.0.10	Switch 1	192.168.0.1	COMPUTER
Client2	eth0	C8:1F:66:07:0F:1D	192.168.0.20	Switch 1	192.168.0.1	COMPUTER
Client3	eth0	C8:1F:66:0F:D6:B9	192.168.0.30	Switch 1	192.168.0.1	COMPUTER
Client4	eth0	D8:CB:8A:60:4C:3B	192.168.0.70	Switch 2	192.168.0.65	COMPUTER
Client5	eth0	D8:CB:8A:60:4B:E0	192.168.0.80	Switch 2	192.168.0.65	COMPUTER
Client6	eth0	C8:1F:66:07:10:7E	192.168.0.90	Switch 2	192.168.0.65	COMPUTER
Printer2		00:15:99:D5:5D:34	192.168.0.100	Switch 2	192.168.0.65	PRINTER
Web Server	eth0	00:1A:A0:56:6C:B2	192.168.0.130	192.168.0.129	192.168.0.129	COMPUTER
</HOSTS AND CLIENT DEVICES>						
<LAYER 2 SWITCHES>						
Switch Name	**Number of connections**					
Switch 1	4					
Switch 2	5					
</LAYER 2 SWITCHES>						

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. INVALID NETWORK PLAN



<NETWORKS>						
Subnet ID	**Notes**					
192.168.0.0/26						
192.168.0.64/26						
192.168.0.128/30						
192.168.0.200/30						
192.168.0.204/30						
192.168.0.208/30						
192.168.0.2122/30						
192.168.0.216/30						
</NETWORKS>						
<ROUTERS>						
Router Name	**Interface Name**	**Interface MAC**	**Interface IP**	**Next Hop (IP or switch name)**		
R1	fa0/0	00:15:63:40:83:80	192.168.0.201	192.168.0.202		
R1	fa0/1	00:15:63:40:83:81	192.168.0.217	192.168.0.218		
R1	fa0/0/0	54:75:D0:7C:70:9F	192.168.0.129	192.168.0.130		
R2	fa0/0	E0:5F:B9:58:8B:74	192.168.0.1	Switch 1		
R2	fa0/1	00:1B:D4:EF:5C:89	192.168.0.209	192.168.0.210		
R2	fa0/0/0	00:1B:D4:EF:5C:88	192.168.0.206	192.168.0.205		
R3	fa0/0	00:1B:54:A9:6D:C8	192.168.0.202	192.168.0.201		
R3	fa0/1	00:1B:54:A9:6D:C9	192.168.0.210	**192.168.0.211**		
R3	fa0/0/0	58:8D:09:76:E9:F4	192.168.0.65	**Switch 3**		
R4	fa0/0	00:18:73:1D:27:00	192.168.0.213	192.168.0.214		
R4	fa0/1	00:18:73:1D:27:01	192.168.0.218	192.168.0.217		
R5	fa0/0	00:17:59:71:34:B0	192.168.0.205	192.168.0.206		
R5	fa0/1	00:17:59:71:34:B1	192.168.0.214	192.168.0.213		
</ROUTERS>						
<HOSTS AND CLIENT DEVICES>						
Host Name	**Interface Name**	**Interface MAC**	**Interface IP**	**Next Hop (IP or switch name)**	**Gateway Address**	**Device Type**
Client1	eth0	C8:1F:66:07:12:05	192.168.0.10	Switch 1	192.168.0.1	COMPUTER
Client2	eth0	C8:1F:66:07:0F:1D	192.168.0.20	Switch 1	192.168.0.1	COMPUTER
Client3	eth0	C8:1F:66:0F:D6:B9	192.168.0.30	Switch 1	192.168.0.1	COMPUTER
Client4	eth0	D8:CB:8A:60:4C:3B	192.168.0.70	Switch 2	192.168.0.65	COMPUTER
Client5	eth0	D8:CB:8A:60:4B:E0	192.168.0.80	Switch 2	192.168.0.65	COMPUTER
Client6	eth0	C8:1F:66:07:10:7E	192.168.0.90	Switch 2	192.168.0.65	COMPUTER
Printer2		00:15:99:D5:5D:34	192.168.0.100	Switch 2	192.168.0.65	PRINTER
Web Server	eth0	00:1A:A0:56:6C:B2	192.168.0.130	192.168.0.129	192.168.0.129	COMPUTER
</HOSTS AND CLIENT DEVICES>						
<LAYER 2 SWITCHES>						
Switch Name	**Number of connections**					
Switch 1	4					
Switch 2	5					
</LAYER 2 SWITCHES>						

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. INCORRECT DETAILS NETWORK PLAN

<NETWORKS>

Subnet ID	Notes
192.168.0.0/26	
192.168.0.64/26	
192.168.0.128/30	
192.168.0.200/30	
192.168.0.204/30	
192.168.0.208/30	
192.168.0.212/30	
192.168.0.216/30	

</NETWORKS>

<ROUTERS>

Router Name	Interface Name	Interface MAC	Interface IP	Next Hop (IP or switch name)
R1	fa0/0	00:15:63:40:83:80	192.168.0.201	192.168.0.202
R1	fa0/1	00:15:63:40:83:81	192.168.0.217	192.168.0.218
R1	fa0/0/0	54:75:D0:7C:70:9F	192.168.0.129	192.168.0.130
R2	fa0/0	E0:5F:B9:58:8B:74	192.168.0.1	Switch 1
R2	fa0/1	00:1B:D4:EF:5C:89	192.168.0.209	192.168.0.210
R2	fa0/0	00:1B:D4:EF:5C:88	192.168.0.206	192.168.0.205
R3	fa0/0/0	00:1B:54:A9:6D:C8	192.168.0.202	192.168.0.201
R3	fa0/1	00:1B:54:A9:6D:C9	192.168.0.210	**192.168.0.211**
R3	fa0/0/0	58:8D:09:76:E9:F4	192.168.0.65	Switch 2
R4	fa0/0	00:18:73:1D:27:00	192.168.0.213	192.168.0.214
R4	fa0/1	00:18:73:1D:27:01	192.168.0.218	192.168.0.217
R5	fa0/0	00:17:59:71:34:B0	192.168.0.205	192.168.0.206
R5	fa0/1	00:17:59:71:34:B1	192.168.0.214	192.168.0.213

</ROUTERS>

<HOSTS AND CLIENT DEVICES>

Host Name	Interface Name	Interface MAC	Interface IP	Next Hop (IP or switch name)	Gateway Address	Device Type
Client1	eth0	C8:1F:66:07:12:05	192.168.0.10	Switch 1	192.168.0.1	COMPUTER
Client2	eth0	**C8:1F:66:07:0F:1E**	192.168.0.20	Switch 1	192.168.0.1	COMPUTER
Client3	eth0	C8:1F:66:0F:D6:B9	**192.168.0.33**	Switch 1	192.168.0.1	COMPUTER
Client4	eth0	D8:CB:8A:60:4C:3B	192.168.0.70	Switch 2	192.168.0.65	COMPUTER
Client5	eth0	D8:CB:8A:60:4B:E0	**192.168.1.88**	Switch 2	192.168.0.65	COMPUTER
Client6	eth0	C8:1F:66:07:10:7E	192.168.0.90	Switch 2	192.168.0.65	COMPUTER
Printer2		**00:15:99:D5:5D:33**	192.168.0.100	Switch 2	192.168.0.65	PRINTER
Web Server	eth0	00:1A:A0:56:6C:B2	192.168.0.130	192.168.0.129	192.168.0.129	COMPUTER

</HOSTS AND CLIENT DEVICES>

<LAYER 2 SWITCHES>

Switch Name	Number of connections
Switch 1	4
Switch 2	5

</LAYER 2 SWITCHES>

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G. INVALID CONFIGURATION NETWORK PLAN

<NETWORKS>

Subnet ID	Notes
192.168.0.0/26	
192.168.0.64/26	
192.168.0.128/30	
192.168.0.200/30	
192.168.0.204/30	
192.168.0.208/29	
192.168.0.216/30	

</NETWORKS>

<ROUTERS>

Router Name	Interface Name	Interface MAC	Interface IP	Next Hop (IP or switch name)
R1	fa0/0	00:15:63:40:83:80	192.168.0.201	192.168.0.202
R1	**fa0/1**	00:15:63:40:83:81	192.168.0.217	192.168.0.218
R1	fa0/0/0	54:75:D0:7C:70:9F	192.168.0.129	192.168.0.130
R2	fa0/0	00:1B:D4:EF:5C:88	192.168.0.1	Switch 1
R2	fa0/1	00:1B:D4:EF:5C:89	**192.168.0.209**	192.168.0.210
R2	fa0/0/0	E0:5F:B9:58:8B:74	192.168.0.206	192.168.0.205
R3	fa0/0	00:1B:54:A9:6D:C8	192.168.0.202	192.168.0.201
R3	fa0/1	00:1B:54:A9:6D:C9	**192.168.0.210**	192.168.0.209
R3	fa0/0/0	58:8D:09:76:E9:F4	192.168.0.65	Switch 2
R4	fa0/0	00:18:73:1D:27:00	**192.168.0.213**	192.168.0.214
R4	fa0/1	00:18:73:1D:27:01	192.168.0.218	192.168.0.217
R5	fa0/0	00:17:59:71:34:B0	192.168.0.205	192.168.0.206
R5	fa0/1	00:17:59:71:34:B1	**192.168.0.214**	192.168.0.213

</ROUTERS>

<HOSTS AND CLIENT DEVICES>

Host Name	Interface Name	Interface MAC	Interface IP	Next Hop (IP or switch name)	Gateway Address	Device Type
Client1	eth0	C8:1F:66:07:12:05	192.168.0.10	Switch 1	192.168.0.1	COMPUTER
Client2	eth0	C8:1F:66:07:0F:1D	192.168.0.20	Switch 1	192.168.0.1	COMPUTER
Client3	eth0	C8:1F:66:0F:D6:B9	192.168.0.30	Switch 1	192.168.0.1	COMPUTER
Client4	eth0	D8:CB:8A:60:4C:3B	192.168.0.70	Switch 2	192.168.0.65	COMPUTER
Client5	eth0	D8:CB:8A:60:4B:E0	192.168.0.80	Switch 2	192.168.0.65	COMPUTER
Client6	eth0	C8:1F:66:07:10:7E	192.168.0.90	Switch 2	192.168.0.65	COMPUTER
Printer2		00:15:99:D5:5D:34	192.168.0.100	Switch 2	192.168.0.65	PRINTER
Web Server	eth0	00:1A:A0:56:6C:B2	192.168.0.130	192.168.0.129	192.168.0.129	COMPUTER

</HOSTS AND CLIENT DEVICES>

<LAYER 2 SWITCHES>

Switch Name	Number of connections
Switch 1	4
Switch 2	5

</LAYER 2 SWITCHES>

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] D. C. McBride, “Mapping, awareness, and virtualization network administrator's training tool (MAVNATT) architecture and framework,” M.S. thesis, Dept. of Computer Science, Naval Postgraduate School, Monterey, CA, 2015.

[11] Sun Microsystems. (2000). *System Administration Guide* [eBook version]. [Online]. Available: https://docs.oracle.com/cd/E19455-01/806-0916/6ja85398m/index.html

[3] Zeezoh95. “*The OSI model...dissected.*” [Online]. Available: https://itsjustusdeveloperstutorialservices.wordpress.com/author/zeezoh95/. [4 September 2015].

[4] IEEE Standards Association. “*Guidelines for 48-bit global identifier (EUI-48).*” [Online]. Available: https://standards.ieee.org/develop/regauth/tut/eui48.pdf. [Accessed 29 March 2016].

[5] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, *RFC 1918: Address Allocation for Private Internets*, 1996.

[6] V. Fuller and T. Li, *RFC 4632: Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan*, 2006.

[7] C. M. Kozierok. “*The TCP/IP guide - IP 'supernetting': classless inter-domain routing (CIDR) hierarchical addressing and notation.*” [Online]. Available: http://www.tcpipguide.com/free/t_IPSupernettingClasslessInterDomainRoutingCIDRHiera-2.htm. [20 September 2005].

[8] Microsoft. “*The TCP/IP model: TCP/IP.*” [Online]. Available: *https://technet.microsoft.com/en-us/library/cc786900%28v=ws.10%29.aspx?f=255&MSPPError=-2147217396.* [21 January 2005].

[9] The Internet Society. “*About the IETF.*” [Online]. Available: https://www.ietf.org/about/. [18 September 2014].

[10] T. Pickett. “*Back to basics with the OSI model - adventures in network engineering.*” [Online]. Available: http://www.tonypickett.com/2013/07/back-to-basics-with-the-osi-model/. [12 July 2013].

[11] K. Wehrle, F. Pahlke, H. Ritter, D. Muller, and M. Bechler. (2004). “Address resolution protocol (ARP),” in *The Linux® Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel* [eBook version]. [Online]. Available: http://flylib.com/books/en/3.475.1.75/1/

[12] K. Wehrle, F. Pahlke, H. Ritter, D. Muller, and M. Bechler. (2004). “The Internet protocol v4,” in *The Linux® Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel* [eBook version]. [Online]. Available: http://flylib.com/books/en/3.475.1.70/1/

[13] G. Lyon. (2011). *Nmap Network Scanning* [eBook version]. [Online]. Available: http://nmap.org/book/

[14] K. Wehrle, F. Pahlke, H. Ritter, D. Muller, and M. Bechler. (2004). “Internet control message protocol (ICMP),” in *The Linux® Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel* [eBook version]. [Online]. Available: http://flylib.com/books/en/3.475.1.74/1/

[15] K. Wehrle, F. Pahlke, H. Ritter, D. Muller, and M. Bechler. (2004). “Transmission control protocol (TCP),” in *The Linux® Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel* [eBook version]. [Online]. Available: http://flylib.com/books/en/3.475.1.118/1/

[16] J. O'Madadhain, D. Fisher, and S, White. (2010). *JUNG - Java Universal Network/Graph Framework* [Online]. Available: http://jung.sourceforge.net/download.html. [Accessed 10 March 2016].

[17] G. Bernstein. “*JUNG 2.0 tutorial.*” [Online]. Available: http://www.grotto-networking.com/JUNG/JUNG2-Tutorial.pdf. [22 April 2009].

[18] U. Brandes, M. Eiglsperger, and J. Lerner. “*GraphML primer.*” [Online]. Available: http://graphml.graphdrawing.org/primer/graphml-primer.html. [Accessed 1 April 2016].

[19] W3C. “*Extensible markup language (XML) 1.0 (fifth edition).*” [Online]. Available: https://www.w3.org/TR/xml/. [26 November 2008].

[20] Oracle, Redwood Shores, CA. (2010). *Java Platform, Standard Edition* [Online]. Available: http://www.oracle.com/technetwork/java/javase/downloads/index.html. [Accessed 9 March 2016].

[21] Object Management Group. “*What is UML | unified modeling language.*” [Online]. Available: http://www.uml.org/what-is-uml.htm. [July 2005].

[22] E. Berndt. “Mapping, awareness, and virtualization network adminstrator training tool: virtualization module,” M.S. thesis, Cyber Academic Group, Naval Postgraduate School, Monterey, CA, 2016.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California